UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO DE FIN DE GRADO

Desarrollo de un algoritmo ACO para el problema de la detección de comunidades en redes sociales

Gianmarco Minelli Sierra

Tutor: Antonio González Pardo

Ponente: David Camacho Fernández

Junio 2019

DESARROLLO DE UN ALGORITMO ACO PARA EL PROBLEMA DE LA DETECCIÓN DE COMUNIDADES EN REDES SOCIALES

AUTOR: Gianmarco Minelli Sierra

TUTOR: Antonio González Pardo

Dpto. Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio de 2019

Resumen (Castellano)

El rápido aumento de la popularidad y el tamaño de las redes sociales en los últimos años, esta área de trabajo se ha convertido en una de las más atractivas para la investigación de la comunidad científica actual.

Ligado al constante crecimiento de las redes sociales se ha creado indirectamente una necesidad de administrar y analizar toda la información que podemos extraer de las mismas. Esta necesidad ha hecho nacer nuevas técnicas de análisis y gestión de datos que nos ofrecen información sobre la estructura, las características y la distribución de la red.

Una de las principales características que nos ofrecen las redes sociales es la posibilidad de relacionarnos con el resto de los usuarios formando de esta manera grupos sociales o comunidades. Estas comunidades pueden tener diferentes raíces, por ejemplo, compañeros de trabajo, amigos del colegio, amigos de la universidad, familiares etc.

El propósito de este Trabajo de Fin de Grado es aportar una solución problema de la detección de estas comunidades en las redes ego de los usuarios de una red social. Un problema de redes de este tipo se puede presentar como un problema de grafos, por lo que utilizaremos un grafo que represente nuestra red ego. El nodo 'ego' será el sujeto bajo estudio, el resto de nodos serán otros usuarios y las conexiones entre ellos representarán su relación de amistad.

Las redes ego que estudiaremos en este proyecto representan una pequeña sección real de la red de Facebook.

El algoritmo desarrollado se clasifica bajo la categoría de los algoritmos de enjambre, ya que está inspirado en el algoritmo de la colonia de hormigas (ACO) donde las hormigas dejan un rastro de feromonas en los caminos recorridos entre una fuente de alimentos y la colonia. En nuestro algoritmo daremos un nuevo enfoque a este comportamiento de las hormigas para poder resolver el problema de detección de comunidades.

Abstract (English)

Due to the increasing growth and popularity of social networks during these last years, this work area has become one of the most attractive areas among the scientific community.

Linked to this increase of social networks, a need to manage and analyse all information that we can extract has been indirectly created. This need has led to the birth of new data management techniques that provide us important information about the structure, characteristics and distribution of the network.

One of the main features that social network offer us is the possibility to interact with other users, thus forming social groups or communities. These communities usually share some characteristics, for example Co-workers, childhood friends, university friends, family etc.

The purpose of this Bachelor Thesis is to provide a solution to the community finding problem in the user's ego networks inside the social network. Usually, these network problems can be represented as Graphs problems, this is the reason why we have decided to implement a graph to depict our ego networks. Ego node corresponds to the subject under study, the rest of nodes corresponds to the different users over the network and connections between nodes represents friendship relation among the users.

The ego networks used in this project are only a slight section of the Facebook's social network.

The developed algorithm can be categorized under the category of swarm intelligence because it is inspired in the ant colony optimization algorithm (ACO) where ants segregate pheromones over the paths taken on their food source searching process. With our algorithm, we will provide a new approach for this ant's behaviour due to resolve the community finding problem over social networks.

Palabras clave (Castellano)

Redes Sociales, algoritmo de colonia de hormigas, comunidad, clúster, heurística, fitness, grafo, inteligencia de enjambre, selección, repetición, iteración, hormiga, feromonas, normalización, red ego.

Keywords (English)

Social networks, Ant Colony optimization algorithm, community finding, cluster, heuristic, fitness, swarm intelligence, selection, repetition, iteration, ant, pheromones, normalization, ego network

Agradecimientos

Me gustaría agradecer este trabajo a mis padres y a mis abuelos que tanto me ha apoyado desde el principio, a mi novia que me ha hecho sacar lo mejor de mi mismo en esta última etapa de la carrera y a mi compañero Darío por tantos buenos momentos a lo largo de estos años de universidad.

Por último, también quiero agradecer este trabajo a mi tutor Antonio González por su ayuda y ánimo.

Índice de contenidos

INDICE DE FIGURAS

Índice Fórmulas

Índice de tablas

Índice de gráficos

1 Introducción

1.1 Motivación

Durante los últimos años se han llevado a cabo numerosos avances tecnológicos que repercuten directamente en la sociedad, uno de los más significativos ha sido el desarrollo de las redes sociales.

La idea de red social es mucho más antigua que el concepto al que estamos acostumbrados en la era digital. Las redes sociales clásicas dependen de una dimensión espacial y del encuentro cara a cara con el interlocutor, mientras que las redes sociales virtuales poseen una lógica distinta y unas dimensiones mucho mayores que las clásicas. En el siglo XX ya se utilizaba el término "red social" para analizar interacciones entre individuos, organizaciones o incluso sociedades. Desafortunadamente, aunque las bases para su estudio ya estaban planteadas, las limitaciones tecnológicas hicieron imposible su avance.

En los últimos años las redes sociales digitales se han introducido en nuestras vidas modificando la forma de relacionarnos con el medio. Desde sus inicios en 1971 gracias a Ray Tomlinson, quien implementó el primer sistema de correo electrónico, las redes sociales han evolucionado hasta convertirse en una herramienta indispensable en nuestra vida social. Sus propias características han permitido su abordaje desde diferentes campos de estudio, desde las matemáticas hasta la sociología, lo que les otorga un valor altamente interdisciplinar. El auge de esta nueva forma de socializar ha hecho emerger la conocida como antropología digital, que crea puentes entre la vida digital de los individuos y el mundo no digital.

La cada vez más alta participación de la población debido a la capacidad de poder conectarse con otras personas en cualquier parte del globo ha hecho que el total de usuarios que utilizan estas redes sociales ascienda a la escalofriante cifra de 2,68 billones de usuarios.

Siendo las redes sociales hogar de tan elevada magnitud de individuos, colectivos, empresas o incluso corporaciones se ha creado indirectamente una necesidad de gestionar toda la información relacionada con ellos. Esta necesidad ha hecho brotar nuevas técnicas de análisis de datos de las redes sociales ofreciéndonos información sobre la estructuración o incluso la distribución de la información a través de estas.

Si nos atrevemos a aproximar el concepto de "red social" a un marco más acotado, podemos imaginarla como una maraña de nodos y lazos conectando dichos nodos.

Los nodos representan a los sujetos a estudiar, que pueden ser personas, grupos, comunidades o corporaciones, y lazos entre los nodos representan los tipos de relación entre ellos.

Estos lazos, por otra parte, pueden ser, entre otros, de semejanza (los que se desarrollan entre nodos de un mismo tipo de atributos: sexo, edad, nacionalidad, etc.), lazos de relaciones sociales (amigos, familiares, etc.) o vínculos de interacciones mediados por pautas de comportamiento (conversaciones, invitaciones, etc.)

En este Trabajo de Fin de Grado se enfocará el análisis del problema de las redes sociales desde el punto de vista de los lazos como relaciones sociales. Definimos la red social como un enorme grafo [1] G = (V, E) siendo V un conjunto finito, donde los nodos (V) representan los usuarios de la red social bajo análisis y los lazos del grafo (E) trazan las relaciones sociales ente los usuarios acordes a la red social. Por ejemplo, si tomamos bajo estudio la red social Facebook, se representaría como un grafo no dirigido en el que los lazos refieren valores de amistad usuario a usuario. De diferente manera, si la red social a estudio fuese Twitter, se representaría como un grafo dirigido en el que los lazos simbolizan el hecho de que un usuario "siga" a otro, pero no implica reciprocidad.

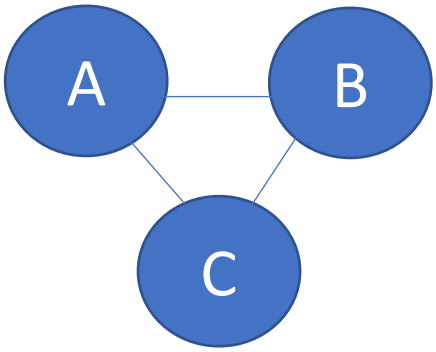


Figura 1: FACEBOOK

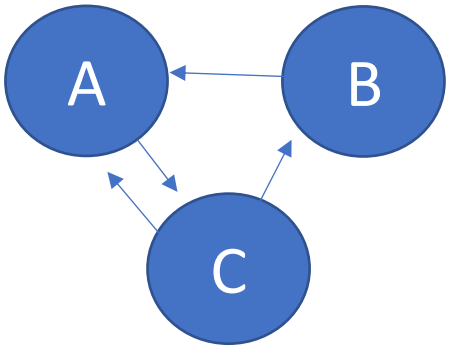


Figura 2: TWITTER

El algoritmo que se ha desarrollado para analizar las posibles comunidades dentro de estas redes sociales se cimienta sobre el trabajo realizado por los investigadores italianos Marco Dorigo y Gianni Di Caro [1] sobre el algoritmo de colonia de hormigas (ACO) y lo aplicaremos al ámbito de las redes sociales que como hemos explicado previamente se interpreta como un problema de grafos. Se desarrollará en el lenguaje de programación Python.

Finalmente utilizaremos como banco de pruebas para el algoritmo un dataset de información real de Facebook, en concreto un sector de la red proporcionado por StandFord University [2] (Todos los usuarios del dataset están cifrados por lo que se respeta una total confidencialidad). Utilizaremos en particular estos datos ya que han sido previamente estudiados por lo que podremos saber de antemano las comunidades existentes en la red para poder verificar el correcto funcionamiento de nuestro algoritmo.

1.2 Objetivos

Este Trabajo de Fin de Grado tiene como objetivo crear y aplicar un algoritmo bio-inspirado de la familia de los algoritmos de enjambre, en particular el algoritmo de la colonia de hormigas (Ant Colony Optimization - ACO) al problema de detección de comunidades en las redes sociales. Utilizaremos este algoritmo sobre un grafo bidireccional que representará una pequeña muestra de la red social Facebook.

El gran objetivo del proyecto es aportar por pequeño que sea nuestro granito de arena al gran trabajo que hay detrás de la comprensión de grandes estructuras digitales enormes tales como las redes sociales. El tratar y analizar toda la información que podemos obtener de estas es vital a la hora de entender la sociedad actual.

Otro de los objetivos de este Trabajo de Fin de Grado es otorgar una nueva aplicación al estudio actual del algoritmo de colonia de hormigas cuyas principales aplicaciones se centran en la búsqueda de caminos más cortos persiguiendo la optimización del tiempo de computación de los problemas bajo el análisis del algoritmo.

1.3 Organización de la memoria

- **Capítulo 2: Estado del arte**

  En este apartado se proporcionará información importante sobre estudios ya existentes relacionados con nuestro propio campo de trabajo y otros próximos. Hablaremos sobre detección de comunidades, clustering, inteligencia computacional, algoritmo de la colonia de hormigas y redes ego.

- **Capítulo 3: Diseño**

  En el capítulo de diseño pondremos las bases sobre el algoritmo propuesto para solucionar el problema de la detección de comunidades en las redes sociales. Daremos una idea general del algoritmo, explicaremos las razones para adaptar el algoritmo ACO a nuestras necesidades, explicaremos el diseño del algoritmo y definiremos nuestras funciones heurísticas, de fitness y por supuesto el criterio utilizado para realizar la asignación de comunidades.

- **Capítulo 4: Desarrollo**

  En este apartado profundizaremos en el algoritmo creado. Explicaremos el entorno de desarrollo que hemos utilizado para su elaboración y el por qué se ha elegido dicho entorno y de la misma manera se explicará el dataset elegido para probar el algoritmo de detección de comunidades. En esta sección también daremos descripciones técnicas de las partes más importantes del algoritmo como la

estructura de ficheros del proyecto, el recorrido que realiza la hormiga por el grafo, el proceso de cálculo de la heurística, el tratamiento de feromonas, el proceso de selección de comunidad, así como el cálculo de fitness de una solución y posteriormente se mostrará un ejemplo paso a paso del camino de una hormiga por el grafo.

- **Capítulo 5: Integración de pruebas y resultados**

  En este capítulo pondremos a prueba el algoritmo desarrollado. Haremos pruebas con diferentes tamaños de repeticiones, iteraciones, hormigas, valores de α y de β. Finalmente juntaremos todas las pruebas que realicemos obteniendo los mejores resultados y discutiremos sobre ellos.

- **Capítulo 6: Conclusiones y trabajo futuro**

  Este será el último capítulo del trabajo donde se expondrán las conclusiones y se reflexionará sobre el proceso de desarrollo del algoritmo y sobre los resultados obtenidos de este. Finalmente se expondrán posibles trabajos futuros que podrán surgir en los próximos años dentro de este marco de los algoritmos bio-inspirados.

Las últimas hojas de este documento están reservadas para las referencias, el glosario donde se encontrarán los términos más técnicos, siglas y abreviaturas y finalmente los anexos.

2 Estado del arte

Como se ha introducido con anterioridad, las redes sociales nos proporcionan la posibilidad de analizar una gran cantidad de datos que pueden ser beneficiosos para entender mejor la estructura y las relaciones de la sociedad. Esto ha hecho que muchas áreas de trabajo como Big Data, estadística, física o aprendizaje automático entre otras hayan enfocado sus esfuerzos en el análisis de las redes sociales. Todos ellos han hecho aportaciones con sistemas, algoritmos o nuevos métodos de obtención de datos.

En este capítulo se hablará sobre trabajos importantes relacionados con la detección de comunidades (Community Finding), algoritmos de enjambre (Swarm Intelligence), algoritmo de la colonia de hormigas (Ant Colony optimization, ACO), clustering y redes ego (Ego Networks)

2.1 Detección de comunidades

La detección de comunidades es una de las piezas más importantes a la hora de realizar un análisis sobre una red. Este trabajo se conecta directamente con la teoría de grafos, reflejando la red como un gran grafo donde los nodos representan los sujetos bajo estudio y los lazos las conexiones que tienen entre sí. El objetivo de este trabajo es agrupar estos sujetos en comunidades (también llamadas "*clusters*" o "*circles*" de los cuales hablaremos más adelante) de manera de que los miembros integrantes de cada comunidad compartan ciertas características.

A la hora de analizar una red, distinguimos dos tipos diferentes de agrupaciones:

- **Agrupaciones de comunidad única**: en estas agrupaciones un sujeto solo puede pertenecer a una comunidad.
- **Agrupaciones de comunidades superpuestas**: en este otro tipo de agrupación un sujeto puede pertenecer a más de una comunidad al mismo tiempo.

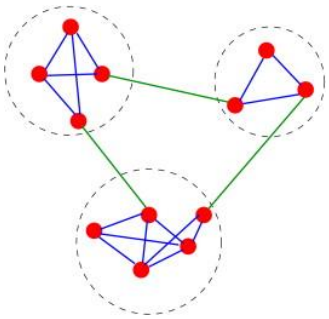


Figura 3: Única comunidad

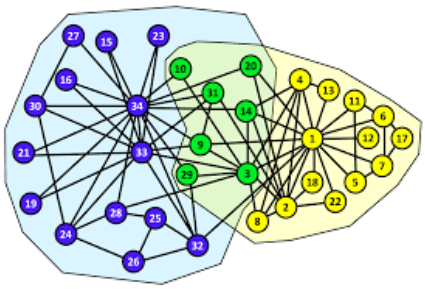


Figura 4: Superposición comunidades

Estas comunidades adquieren esta morfología debido a que los nodos comparten ciertas propiedades, relaciones o lazos entre ellos, por lo tanto, los nodos de las diferentes comunidades suelen tener alguna característica común que no se debería ver en otras comunidades. De este modo se llegó a la conclusión de que existen dos tipos relevantes de información útil que se pueden usar a la hora de hacer un análisis de una red, siendo muy importante elegir bien el tipo de información que se utilizará pues cada uno tiene sus ventajas y sus desventajas. Estos dos tipos de información que se encuentran son:

- **Información asociada a los nodos de la red:** esta información es muy útil si el estudio que se quiere llevar a cabo sobre la red está basado en las propias características de los nodos, sin embargo, tendrá problemas para clasificar el nodo en caso de que tenga poca información característica propia.
- **Información asociada a las conexiones de la red:** este otro tipo de información nos será valiosa si el estudio de la red es referente a las relaciones que tengan los sujetos de la red entre sí, la parte negativa la encontramos cuando se pretende clasificar nodos que tengan pocas conexiones con otros nodos de la red.

Por tanto, es comprensible que a medida que la red aumente, lo hagan de forma habitual también las comunidades, dado, por un lado, aparecen nuevas características de los propios nodos redefiniendo las comunidades y, por otro lado, el incremento de nodos hace que se creen más conexiones con los nodos de la red original, dando lugar a comunidades más grandes o nuevas comunidades. Esto hace más ardua la tarea de particionar el grafo en comunidades de forma óptima y es la mayor dificultad que podemos encontrar en el problema de partición de la red en comunidades.

Lo que normalmente se espera a la hora de crear una comunidad, es que los nodos incluidos en esta tengan una gran cantidad de conexiones entre sí (Conexiones Intra-Comunidad) lo que implicaría que los nodos de la comunidad están fuertemente relacionados o que comparten las mismas características. Por otro lado, también es deseable que los nodos de las comunidades tengan las mínimas conexiones con nodos de otras comunidades (Conexiones Inter-Comunidad) significando esto que las comunidades tienen sus propias características y que la relación entre ellas es la mínima posible.

En la actualidad existen diferentes algoritmos creados específicamente para el problema de detección de comunidades. Entre ellos destacan algoritmos de la rama de métodos jerárquicos como el de Newman y Girvan [3], y de otros no jerárquicos como por ejemplo modulares o particionales (todos estos algoritmos se describirán brevemente en el próximo punto)

2.2 Clustering

Haciendo referencia al apartado anterior, los problemas de detección de comunidades están directamente relacionados con los problemas de clustering cuyo principal objetivo es agrupar la información en clusters. Los problemas de clustering son parte de una

importante área del aprendizaje no supervisado (Unsupervised Learning) que se caracteriza por tratar los objetos de entrada como un conjunto de variables aleatorias en los que la red descubre de forma autónoma características y correlaciones entre estas. A diferencia del aprendizaje supervisado suelen requerir menos tiempo de entrenamiento.

Existen dos ramas principales de estudio de clustering: las que se enfocan en las medidas de las similitudes entre la entrada de los datos y las que hacen hincapié en el proceso de clustering. Desarrollaremos estas últimas brevemente a continuación.

Disponemos de varios métodos de clustering en función de cómo de compleja sea la red por analizar, uno de los más famosos es el jerárquico (Hierarchical). El objetivo de este método es agrupar clusters para así formar uno nuevo o a partir de un cluster formar dos, es decir, separarlo. Con los nuevos clusters obtenidos se vuelven a realizar los procesos de aglomeración o disociación, que consisten en:

- **Aglomeración**: los métodos aglomerativos (ascendentes) se caracterizan por iniciar el análisis con un cluster para cada nodo de la red. A partir de estos clusters se irán formando nuevos grupos de datos hasta que finalmente todos estén agrupados en una misma comunidad.
- **Disociación**: los métodos disociativos (descendentes) se caracterizan por iniciar el análisis con un cluster general que engloba a todos los nodos de la red. A partir de este, a través de reiteradas divisiones, se van formando grupos de nodos cada vez más pequeños hasta que al final de este proceso se tienen tantos clusters como nodos hubiese en la red.

Estas asociaciones y divisiones suelen ser realizadas por algoritmos avariciosos ("*greedy*") siguiendo una heurística y sus resultados suelen ser representados en un dendrograma. El problema de estos algoritmos es el número concreto de comunidades a determinar, ya que la mayoría de las veces el analizador de la red no está interesado en la jerarquía completa sino solo en una parte de esta, por lo que se decide partir el dendrograma. Uno de los algoritmos más famosos dentro de este campo es el Método de Newman & Girvan [3] que elimina progresivamente los bordes de la red original centrándose en los bordes inter-comunidad.

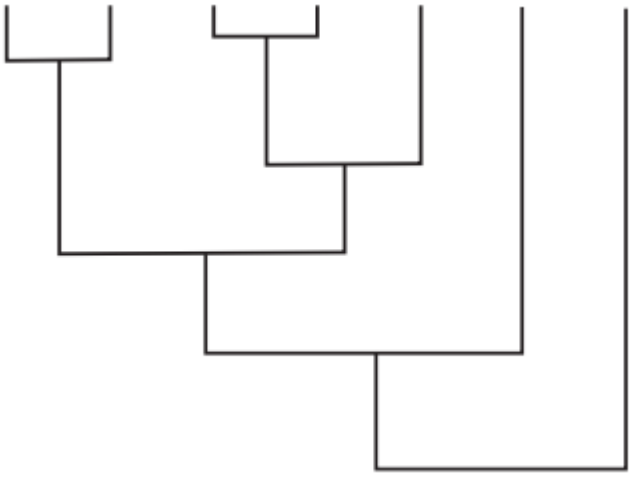


Figura 5: Ejemplo de dendrograma

Otros métodos de clustering importantes son los particionarios. Estos métodos se diferencian de los jerárquicos en que su objetivo es partir la red en K clusters donde K es un valor que se especifica antes del inicio del algoritmo o bien se define como una parte del proceso.

El objetivo de este tipo de técnicas, disponiendo inicialmente de una partición de los individuos hacer que los clusters interaccionen entre ellos recibiendo o mandando individuos a otros clusters, siendo indispensable que el número de clusters K no varíe. Estos cambios de comunidades se hacen con la intención de encontrar una solución mejor de la que se disponía previamente. Los diferentes algoritmos dentro de esta bolsa de métodos particionarios de clustering difieren en lo que entiende cada uno por una "*partición mejor*" y en cómo conseguirlo. Algunos de los algoritmos más famosos que utilizan esta estrategia de particionado de la red son: K-means [4] o K-medoids [5]

En los últimos años han surgido diferentes algoritmos bajo el paradigma de la inteligencia computacional que se caracterizan por basarse en comportamientos biológicos. Por ejemplo, algoritmos de redes neuronales, algoritmos evolutivos, algoritmos de inteligencia de enjambre, algoritmos basados en el sistema inmunológico o algoritmos de agrupamiento difuso (fuzzy systems).

2.3 Inteligencia Computacional

Enlazando con lo que se explicaba en el anterior capítulo, en los últimos años se ha dado un gran avance al desarrollo de nuevos algoritmos para solucionar problemas complejos.

Uniendo la lógica, un razonamiento deductivo, el uso de sistemas expertos y sistemas de aprendizaje automático podemos englobar a estos nuevos algoritmos bajo el campo de la Inteligencia Artificial.

Pero ¿puede una máquina ser inteligente? Esta es una pregunta que todavía sigue generando mucha controversia. A mediados del siglo XX Alan Turing aportó mucho a este debate. Opinaba que se podían crear máquinas que simularan un cerebro humano, que no existía nada que el cerebro pudiese hacer que una máquina bien diseñada no lograse. Y aunque se han hecho grandes descubrimientos y avances en la materia como el modelamiento de redes neuronales, todavía falta una solución exacta al problema de la intuición, la conciencia o las emociones humanas.

En este apartado nos centraremos en la rama de la inteligencia artificial llamada Inteligencia Computacional que se enfoca en el estudio de mecanismos adaptativos que faciliten un comportamiento inteligente en entornos cambiantes y complejos.

En este Trabajo de Fin de Grado hablaremos del avance de algunos de los algoritmos bio-inspirados donde se incluyen redes neuronales inteligentes, computación evolutiva, inteligencia de enjambre, sistemas inmunológicos artificiales o sistemas difusos entre otros.

Mientras que aplicar cada una de estas técnicas de manera individual ya ha dado resultados exitosos a la hora de resolver problemas complejos reales, las nuevas corrientes de estudio pretenden es desarrollar sistemas híbridos, ya que ninguno de ellos es superior a otro en todas las situaciones, buscando así eliminar las debilidades individuales de cada uno y aunar sus puntos fuertes.

A continuación, hablaremos sobre computación evolutiva e inteligencia de enjambre, los paradigmas bajo los cuales se ha desarrollado este Trabajo de Fin de Grado.

2.3.1 Computación Evolutiva

El objetivo principal de la computación evolutiva es la simulación de procesos de evolución natural y su aplicación a la resolución de problemas complejos. LA idea fundamental de estos algoritmos es aplicar los conceptos de evolución introducidos por Charles Darwin en los que prima la supervivencia del fuerte frente a la extinción del débil. En la naturaleza, la evolución se consigue a través de la reproducción y mutación de las especies a lo largo del tiempo. De la evolución se espera que el material genético dado por los progenitores a sus descendientes sea el mejor posible ya que si estos reciben un material genético deficiente, no perdurarán.

Los algoritmos evolutivos toman inicialmente una población de individuos que contienen genes. Lo que se pretende con estos algoritmos es encontrar una solución óptima entre ellos o sus descendientes dada una función de fitness a lo largo de las generaciones. Para ello por cada generación, los individuos pasan por un proceso de reproducción, llamado cruce, en el que intercambian parte de sus características con otros individuos. Como es lógico, y al igual que en la naturaleza, es altamente probable que el cruce de dos individuos con unos muy buenos genes dé lugar a un descendiente con unos genes igual de buenos o incluso mejores. Posteriormente los descendientes

pasan por un proceso de mutación que altera parte de sus características en función de lo que el problema requiera.

Tras cada generación se debe de decidir cuáles de estos nuevos individuos pasan el corte evolutivo, esto se conoce como *elitismo*.

El algoritmo no cesará de producir nuevas generaciones en busca de una solución óptima dada por un valor de fitness que el analizador haya estipulado previamente al inicio del proceso. El algoritmo también puede parar cuando llegue a un número N de generaciones dadas por el analizador y obtener la mejor solución en ese momento de la evolución.

Existen diferentes variantes de computación evolutiva que difieren entre ellos en qué tipo de individuos seleccionar, la forma en que se cruzan los individuos o el método de mutación de los descendientes entre otros.

2.3.2 Inteligencia de Enjambre

Los sistemas de inteligencia de enjambre están inspirados en la naturaleza, especialmente en los sistemas biológicos. Hace referencia a un grupo de técnicas y sistemas que están basadas en el estudio del comportamiento colectivo de sistemas autoorganizados y distribuidos, es decir, descentralizados. De manera usual, estos sistemas se caracterizan por estar formados por una población de agentes capaces de percibir y modificar el ambiente de forma local.

Como mencionamos anteriormente, no existe una estructura centralizada de control que envíe órdenes a los agentes sobre cuál debe ser su comportamiento, sino que son las propias interacciones entre los agentes las que producen una gran cantidad de procesos simples que al unirse producen uno complejo. A esto se le conoce como estigmergia.

Dentro de este marco de trabajo podemos encontrar diferentes técnicas:

- **Optimización de colonia de hormigas (ACO):** inspirada en los comportamientos de las colonias de hormigas cuando buscan un camino entre la colonia y una fuente de alimentos.
- **Optimización de enjambre de partículas (PSO**): inspirados en el comportamiento de las bandadas de aves o los bancos de peces. [6]
- **Algoritmo de colonia de abejas (ABC):** inspirados en el comportamiento inteligente de las colonias de abejas en la búsqueda de miel. [7]
- **Algoritmo de murciélago**: inspirados en los cambios de frecuencia, volumen y pulso que llevan a cabo los murciélagos al volar. [8]

Estos son solo unos ejemplos de las técnicas que existen actualmente, pero hay muchas más.

En este Trabajo de Fin de Grado nuestro objetivo principal es centrarnos en la optimización de colonia de hormigas, aunque como se puede observar hay diversas técnicas exploradas bajo la inteligencia de enjambre.

2.4 Algoritmo de colonia de hormigas

Las hormigas son seres que viven en organizaciones sociales altamente estructuradas. Gracias a esto, son capaces de realizar tareas complejas a través de sus acciones individuales.

Los algoritmos de colonia de hormigas fueron presentados por primera vez por Marco Dorigo en 1992. Este tipo de algoritmos simulan el comportamiento que tienen las hormigas a la hora de recolectar comida de una fuente de alimentos y llevarla de vuelta a la colonia.

Las hormigas salen en busca de comida aleatoriamente, pero en el momento en el que una fuente de comida es localizada, los patrones de actividad de las hormigas que hasta entonces eran arbitrarios pasan a ser más organizados con más y más hormigas siguiendo el camino que conecta con la fuente de comida. Tarde o temprano y como por arte de magia, todas las hormigas acaban siguiendo el mismo camino que resulta ser el más corto. Este comportamiento es consecuencia de que las hormigas que encontraron la fuente de comida se comunican con las hormigas que deben de ir por primera vez a la fuente por lo que tienen influencia sobre ellas. Esta comunicación varía según las especies.

En el caso de las hormigas la forma de comunicarse entre ellas es a través de rastros de feromonas. Cuando una hormiga encuentra un camino que lleva a una fuente de alimentos y vuelve a la colonia deja tras de sí un rastro de feromonas que ha ido desprendiendo en el camino recorrido. ¿Por qué es importante este rastro de feromonas? Cuando una nueva hormiga debe ir a la fuente de alimentos decidirá qué camino elegir en base a la concentración de feromonas que haya en las diferentes vías. Se puede deducir lógicamente que un camino con una concentración mayor de feromonas tiene más posibilidades de ser elegido que uno en el que apenas hay rastro de ellas. El hecho de que más hormigas tomen un mismo camino hará que atraigan a una cantidad mayor de hormigas a este.

Este rastro de feromonas que desprenden las hormigas cuando recorren el camino sufre un proceso de evaporación a lo largo del tiempo. Este hecho es muy importante ya que gracias a él el algoritmo de optimización evita caer en soluciones locales y realiza una exploración más robusta del espacio de soluciones. En el supuesto de que no existiese la evaporación de feromonas, los caminos elegidos por la primera hormiga pasarían a ser mucho más tentadores para el resto de las hormigas. Gracias a la evaporación de feromonas podemos decir que el algoritmo ACO es un algoritmo metaheurístico.

2.4.1 El experimento de puente binario

Este experimento llevado a cabo por Jean-Louis Deneubourg, fue uno de los primeros trabajos realizados para comprender la naturaleza de la toma de decisiones de las hormigas en función de las feromonas que estas emitían [9].

En este experimento se unió una colonia de hormigas con una fuente de alimentos a través de dos caminos separados por exactamente la misma distancia.

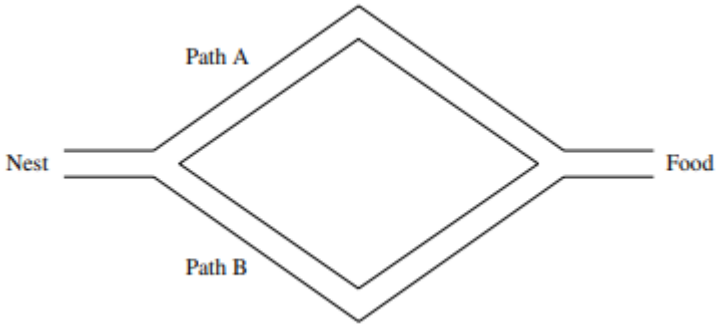


Figura 6: Experimento puente binario

Al inicio, las primeras hormigas escogían de manera arbitraria uno de los caminos por el que ir y volver a la fuente de alimentos. Cada una de estas hormigas iniciales segregaba feromonas que se depositaban en el camino elegido.

¿Qué sucedía con el resto de la colonia? A medida que las hormigas se decantaban por uno de los dos caminos, comenzaba un proceso de retroalimentación provocado por la segregación de feromonas de las hormigas precedentes que hacía que las nuevas hormigas se decidiesen con una probabilidad mayor por el camino que tuviese el rastro de feromonas más potente. Finalmente, todas las hormigas circulaban por el mismo camino.

2.4.2 Experimento de selección del camino más corto

Este otro experimento, llevado a cabo por Goss, fue una extensión del experimento realizado por Deneubourg. En este caso uno de los caminos era más corto que otro a diferencia del experimento anterior donde ambos tenían la misma distancia.

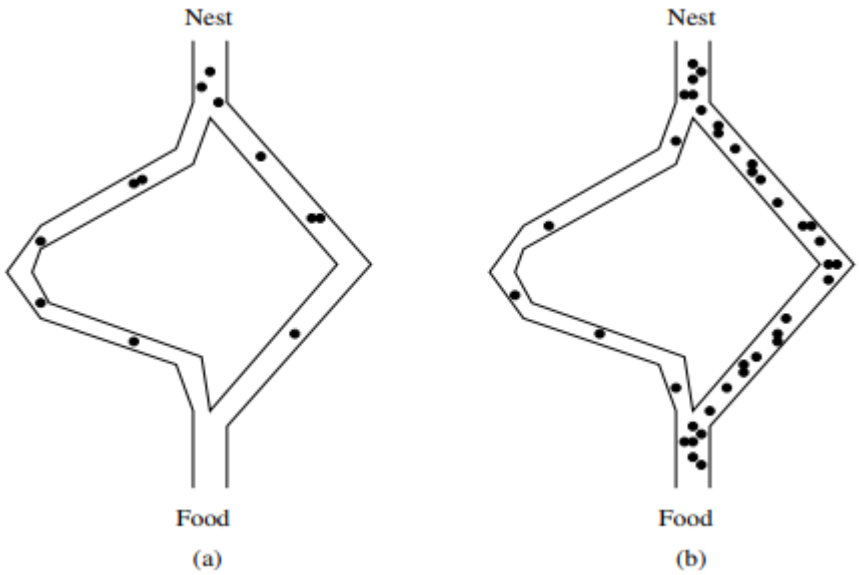


Figura 7: Experimento camino más corto

Al principio las hormigas elegían el camino a la fuente de alimentos de manera aleatoria como en el experimento anterior.

¿Cuál es la diferencia entonces? Pues bien, como es lógico, las hormigas que habían elegido el camino más corto vuelven también por el camino más corto por lo que llegan antes de nuevo a la colonia que las hormigas que tomaron el camino más largo. Esto implica que están dispuestas de nuevo a salir por el mismo camino antes que las otras hormigas por el suyo por lo que al tardar menos tiempo y poder recorrer el camino más veces, el rastro de feromonas del camino corto será más potente que el otro camino, y que a la hora de que se evaporicen las feromonas de los caminos seguirá habiendo más en el camino corto por lo que lo hará más atractivo para el resto de las hormigas.

Finalmente, al igual que en el experimento anterior, todas las hormigas acaban convergiendo en un camino, pero en este caso coincide que siempre convergen al camino más corto. De hecho, el tiempo empleado por las hormigas para convergir en el camino más corto es significativamente menor al tiempo que empleaban las hormigas en el experimento del puente binario donde los caminos tenían la misma distancia a la fuente de comida.

2.5 Redes Ego

El incremento masivo de usuarios en las redes sociales ha hecho que el volumen de información que se puede extraer de ellas sea muy elevado y por lo tanto costoso. Por ello, diferentes trabajos se han enfocado en la obtención de esta información mediante el empleo de redes ego.

Una red ego está compuesta por un nodo que será el centro de la red, denominado “ego”, el resto de los usuarios que estarán unidos con el nodo ego, llamados “alters” y las conexiones entre el resto de alters.

Por lo tanto, para el análisis de una red de N usuarios, existirán N redes ego diferentes, una por cada usuario de la red original.

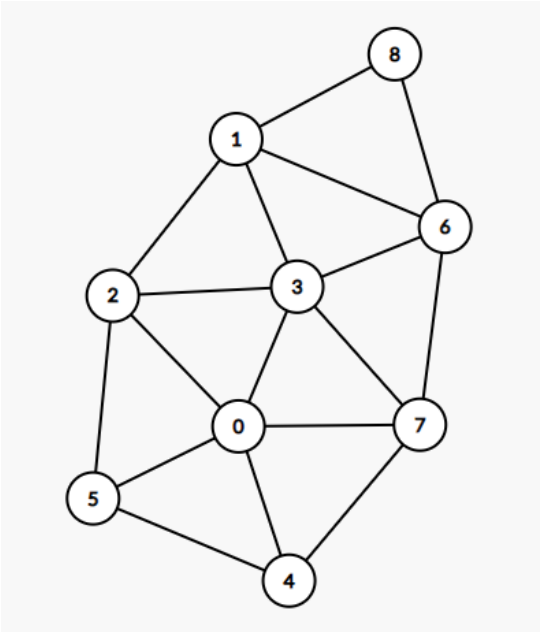


Figura 8: Ejemplo de red social

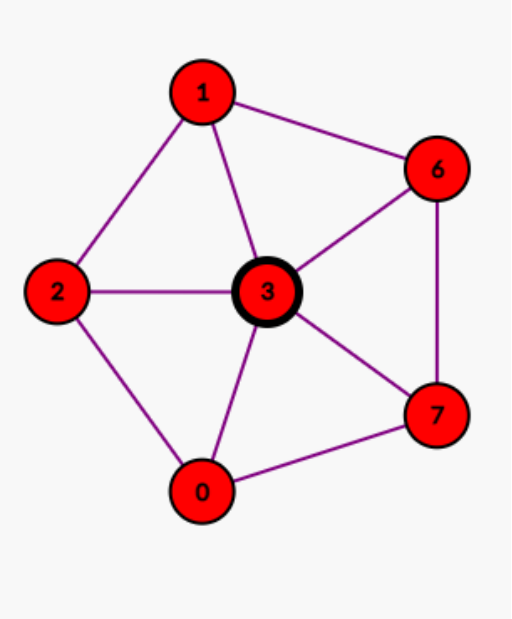


Figura 9: Red ego para el usuario 3

Por lo tanto, las aplicaciones que trabajan con redes ego se centran en encontrar las comunidades de los usuarios individualmente.

Existen diferentes algoritmos que utilizan redes ego para realizar análisis de detección de comunidades:

- **Método de percolación de cliques (CPM):** este método estudia la superposición de comunidades como veíamos en la Figura 4. Este algoritmo busca todos los cliques de tamaño k llamados (k-cliques). Recordemos que un clique es un grafo en el que todos los nodos están conectados entre sí. Normalmente los grafos con los que representamos las redes sociales suelen tener subgrafos que cumplen la condición de clique. Una vez que se encuentran todos los k-cliques del grafo, se reduce el grafo original formando nuevos nodos para que sustituyan a estos k-cliques. Dos de estos nuevos nodos pertenecerán a la misma comunidad si comparten por lo menos k-1 nodos. [11]
- **Algoritmo de propagación de etiquetas:** este algoritmo es también utilizado para detectar comunidades dentro de una red. Se cimienta en la propagación de etiquetas sobre los lazos de la red para definir las comunidades. Al inicio del algoritmo todos los nodos tienen una etiqueta distintiva con la comunidad a la que pertenecen y esta pertenencia a dicha comunidad se ve modificada en función de las etiquetas que poseen los nodos adyacentes. Esto provoca que los nodos que estén fuertemente conectados obtengan una etiqueta común más velozmente. [12]

3 Diseño

3.1 Idea General del algoritmo propuesto

El algoritmo que se ha desarrollado en este Trabajo de Fin de Grado es una ampliación del trabajo realizado por Marco Dorigo sobre el comportamiento las colonias de hormigas en la naturaleza. Por lo tanto, nuestro algoritmo estará basado en el método de optimización de las colonias de hormigas, pero a diferencia del resto de que suelen buscar optimizaciones para hallar caminos óptimos, nuestro proyecto está orientado al área de la detección de comunidades.

El auge de las redes sociales en los últimos años y su gran repercusión sobre la sociedad han motivado la elección de esta área de trabajo. El aumento por lo tanto de información que se puede extraer de estas redes hace esta área de conocimiento más atractiva para los investigadores.

Para la correcta aplicación del algoritmo hemos transformado la red social en un grafo no dirigido en el que los nodos corresponden a los usuarios de la red y las conexiones entre ellos representan si ambos usuarios tienen una relación de amistad.

El tipo de red elegida para hacer los estudios sobre ella será de tipo de red ego para poder centrarnos en un análisis individual de los usuarios pues lo que nos interesa al final es poder detectar las diferentes comunidades para cada usuario en particular.

El concepto del algoritmo ACO original era que las hormigas salían en busca de fuentes de comida y cuando encontraban una, al realizar el camino de vuelta dejaban feromonas en su recorrido para comunicarse con el resto de las hormigas. Estas hormigas posteriormente utilizaban estas feromonas para elegir el camino a tomar, es decir, que las feromonas de las hormigas anteriores influían en su decisión.

En nuestro algoritmo mandaremos hormigas a recorrer el grafo construido en función de la red social bajo estudio de una en una y, en función de ciertos criterios que se detallarán en las secciones siguientes, irá formando comunidades a lo largo de su recorrido.

A diferencia del algoritmo original donde las hormigas dejaban las feromonas en el camino (si lo aplicamos a la teoría de grafos, sería equivalente a depositarlas en las conexiones del grafo), en nuestra implementación del algoritmo cuando la hormiga termina de recorrer todos los nodos.

Por lo tanto, cuando las hormigas que la sucedan visiten los nodos del grafo encontrarán en ellos las feromonas de las hormigas que recorrieron el grafo en el pasado y podrán ajustar su decisión en función de estas feromonas.

Para evitar caer en máximos locales y realizar una exploración de soluciones más robusta, las feromonas pasan por un proceso de evaporación a lo largo del tiempo como en el algoritmo ACO original. En nuestro caso las feromonas se evaporarán al final una iteración de un número N hormigas sobre el grafo.

3.2 Definición del algoritmo

Como mencionábamos en el apartado 3.1, el objetivo de este Trabajo de Fin de Grado es aplicar un algoritmo de enjambre, en particular un algoritmo basado en el algoritmo de Optimización de Colonia de Hormigas (ACO) al problema de la detección de comunidades en redes sociales. La red social elegida para este estudio ha sido Facebook.

Para resolver este problema hemos cambiado el paradigma de ida y vuelta de una fuente de alimentos al paradigma de la teoría de grafos formando un grafo que las hormigas recorrerán por completo e irán formando las respectivas comunidades de un usuario a medida que viajan por él.

Lo primero que debemos realizar es recoger los datos de configuración del algoritmo que serán los siguientes:

- **N.º Repeticiones**: número de repeticiones por las que pasaremos el algoritmo de detección de comunidades sobre el mismo grafo "virgen", es decir, sin ningún rastro de feromonas previo. La primera hormiga que se lanza por el grafo será literalmente la primera que recorra el grafo y deje las primeras feromonas.
- **N.º Iteraciones**: número de veces que se lanza un conjunto de hormigas sobre el grafo. Estos nuevos grupos de hormigas recorren el grafo que contiene el rastro de feromonas de grupos de hormigas anteriores.
- **N.º de hormigas**: número de hormigas que se lanzarán en cada iteración encargadas de recorrer el grafo e ir depositando las feromonas en los nodos mientras crean las comunidades.
- **α**: valor que representa la prioridad que se le querrá dar a la heurística de la solución local del problema a la hora de tomar la decisión de selección de comunidad.
- **β**: valor que representa la prioridad que se le querrá dar a la recogida de feromonas del grafo depositadas por las hormigas predecesoras a la hora de tomar la decisión de selección de comunidad.
- **Evaporation Rate**: valor que representa el ratio de evaporación de las feromonas del grafo tras cada iteración. Este valor tomará valores en el rango [0,1]

Una vez recogidos los parámetros de configuración creamos el grafo a través de los ficheros [nodo].edges del dataset que nos ofrece StandFord University y de la herramienta NetworkX. Explicaremos esto con más detalle en el **apartado 4.2**

Al crear el grafo de esta manera debemos tener cuidado pues el al crear el grafo a través del fichero, no existe el nodo "ego" que sería el nodo que aplica al fichero. Por lo tanto, debemos crear este nodo y crear una conexión desde este a cada uno del resto de nodos del grafo. Esto es imprescindible para que la hormiga consiga recorrer el grafo de manera completa.

IMPORTANTE: el nodo "ego" no se incluirá en ninguna comunidad ya que este es el nodo bajo estudio. Por lo tanto, cuando se llegue a este lo saltaremos. Veremos esto más adelante cuando describamos el movimiento de la hormiga por el grafo.

El siguiente paso que debemos realizar es configurar todos los nodos del grafo. Cada nodo tendrá la siguiente información:

- **Visitado**: variable que controla si el nodo ya ha sido visitado por la hormiga ya que los nodos solo serán visitados una vez porque según las propiedades que hemos elegido para nuestro algoritmo, un nodo solo podrá estar en una comunidad y tan solo una vez. **Ver Figura 3.**
- **Feromonas**: diccionario que mantiene el histórico de feromonas que van dejando las hormigas tras su paso el nodo. La clave del diccionario corresponderá a Tuplas (N_1, N_2) del propio nodo con algún vecino con el que compartiese solución en el pasado y la clave del diccionario será el fitness acumulado de esa solución.

Todos los nodos se inicializarán con valor visitado a 0 excepto el "ego" que se inicializará con valor visitado -1. El diccionario de feromonas se inicializa vacío.

Una vez creado el grafo e inicializado los nodos de este, comenzamos el bucle principal.

1. El algoritmo se repetirá el N.º Repeticiones definido en el fichero de configuración.
2. Dentro de cada repetición se lanzarán baterías de K hormigas tantas como N.º Iteraciones que esté definido en el fichero de configuración.
3. Cada batería de hormigas estará definida como N.º de hormigas en el fichero de configuración de la misma manera.
4. En el inicio del bucle del N.º de hormigas, es decir, para cada hormiga, es donde comienza realmente la funcionalidad.
5. En primera instancia debemos crear las hormigas que portarán con ellas los valores de α y de β. **Ver apartado 3.3**
6. Una vez inicializada nuestra hormiga, empezará su recorrido del grafo. **Ver apartado 4.4**
    - Cada vez que la hormiga pasa por un nodo diferente al nodo "*ego*" se estudia su heurística y se normaliza. **Ver apartados 3.4 y 4.5**
    - Tras haber hallado la heurística de la red, se obtienen las feromonas del nodo. **Ver apartado 4.6**
    - Teniendo en cuenta la heurística (topología de la red) y las feromonas del nodo en el que se encuentra la hormiga elige una comunidad para ese nodo.
    - Cuando finaliza el proceso de selección de comunidad para ese nodo, lo guarda en su solución local.
    - Tras este proceso, marcamos el nodo actual como visitado y lo eliminamos de la lista de nodos que quedan por visitar del grafo.
    - Este proceso se repetirá desde el punto 6 hasta aquí. Finalmente, cuando no queden nodos por visitar en el grafo tendremos formada la solución de la hormiga.

7. Finalizado este proceso, calculamos su fitness y guardamos la solución local de la hormiga. **Ver apartados 3.5 y 4.8**
8. Inicializamos de nuevo los nodos al estado no visitado para que la próxima hormiga pueda recorrer el grafo y aplicamos la evaporación de feromonas en función del factor definido como Evaporation Rate en el fichero de configuración. **Ver apartado 4.6**
9. Una vez todas las hormigas de la misma iteración han pasado por el grafo se selecciona la mejor solución y se guarda.
10. Cuando finalicen todas las iteraciones de las baterías de hormigas y tengamos los resultados de las mejores hormigas de cada batería, obtenemos la mejor y la guardamos nuevamente ya que será el mejor resultado de cada repetición.
11. Este sería el algoritmo, que se repetirá como se mencionaba en el punto 1 en función de N.º de repeticiones. Finalmente obtenemos el mejor resultado de entre todas las repeticiones realizadas y esta será nuestra solución.

3.3 Diseño de la hormiga

La hormiga es la estructura más importante de este algoritmo al ser el centro de toda la lógica que se sigue para desarrollarlo ya que es precisamente, de su comportamiento del que nació la idea de aplicaciones de optimización como la del Algoritmo de Colonia de Hormigas (ACO). Como se menciona en el apartado anterior la hormiga es la encargada de recorrer el grafo y asignar las comunidades a los nodos a medida que los va visitando. Para asignar dichas comunidades la hormiga debe de portar cierta información que junto con los datos de la topología de la red y la información que recoja al llegar al nodo a asignar tomará la decisión de en qué comunidad agrupar este nuevo nodo.

Estos datos que porta la hormiga son:

- **α**: como se explicaba en el **apartado 3.**2 es un valor que representa la prioridad que se le querrá dar a la heurística a la hora de tomar la decisión de selección de comunidad. Se elevará el valor de la heurística en la **Formula 6** a este valor.
- **β**: como se explicaba en el **apartado 3.2** es un valor que representa la prioridad que se le querrá dar a la recogida de feromonas del grafo depositadas por las hormigas predecesoras a la hora de tomar la decisión de selección de comunidad. Se elevará el valor de las feromonas en la **Formula 6** a este valor.
- **statusDicc**: diccionario en el que guardará la solución local de la hormiga paso a paso del problema de detección de comunidades. La clave del diccionario es el id de la comunidad a la que se le corresponderá la lista de nodos pertenecientes a esa comunidad como valor.

3.4 Heurística

Existen diferentes funciones heurísticas para analizar una red. Una de ellas es la que se enfoca en la topología de la propia red, es decir, que tiene en consideración la información que refleja la red como las conexiones entre nodos para llevar a cabo la tarea de búsqueda de comunidades. Esta función heurística es precisamente la que utilizaremos en este Trabajo de Fin de Grado.

La topología de la red es uno de los factores que influirán en la toma de decisión de la hormiga a la hora de seleccionar comunidad como mencionábamos anteriormente.

La función heurística de la topografía de la red al incorporar el nodo **i** a la comunidad j se define como:

$$Topology\left(e_i, C_j\right) = \tau_{ij} = \frac{\left|\mathcal{N}_i \cap C_j\right|}{\mathcal{N}_i}$$

Fórmula 1: Cálculo de Heurística

Donde:

- $\boldsymbol{\mathcal{N}_i}$: son los vecinos del nodo i que estamos estudiando.
- $\boldsymbol{C_j}$: son los nodos pertenecientes a la comunidad j.
- $|\cdot|$: es la función que indica el número de elementos que contiene lo que haya en su interior.

No solo existe la topografía como función heurística, sino que también puede darse una heurística en función de la información almacenada en los perfiles de los usuarios, lo que quiere decir que tomaría en consideración las características de los nodos del grafo.

3.5 Fitness

Como es usual en la mayoría de los algoritmos bio-inspirados, la calidad de las soluciones se suele medir para poder saber el grado de acierto que tienen. Estas mediciones se realizan a través de funciones fitness que son las que evalúan cómo de buena es la solución planteada.

Estas funciones objetivo suelen estar acondicionadas al problema que se desea resolver por lo que serán diferentes dependiendo de lo que se espere del algoritmo. Las funciones fitness independientemente de cuáles sea suelen intentar maximizar (no siempre, puede darse el caso de que se busque minimizarlas) pero por regla general las funciones fitness suelen estar orientadas a tener un valor más elevado cuanto mejor sea una solución.

Para el algoritmo que pretende resolver el problema de detección de redes sociales que se desarrolla en este Trabajo de Fin de Grado, se ha establecido la función fitness según el grado de cohesión de las comunidades obtenidas y la separabilidad entre ellas. Se han elegido estas dependencias del fitness al estar haciendo un estudio sobre la topología de

la red, es decir, sobre las conexiones entre los usuarios como explicábamos en el punto anterior.

- **Cohesión**: definimos la cohesión $\boldsymbol{\delta_{int}}$ como el grado de conectividad de todos los nodos de una comunidad en relación con el tamaño de la comunidad.

$$\delta_{int}(x) = \frac{\#internal\_edges}{\frac{n_x(n_x - 1)}{2}}$$

Fórmula 2: Cálculo de cohesión

Donde:

  - #**internal_edges**: es el número total de conexiones los nodos de la misma comunidad.
  - $\boldsymbol{n_x}$: corresponde al número de nodos de la comunidad x sobre la que se está aplicando la fórmula de cohesión.

Lo que se pretende con esta definición de cohesión es premiar a las comunidades que estén altamente conectadas frente a las que tengan pocas conexiones entre ellas.
Esta función tiene un rango de valores entre [0,1], siendo el valor 0 el peor caso cuando el nodo está solo en la comunidad y el valor de 1 cuando la comunidad forma un clique (o lo que es lo mismo, que todos los nodos estén conectados con todos los nodos).

- **Separabilidad**: definimos la separabilidad δ_{ext} como el grado de conectividad de una comunidad con el resto de las comunidades.

$$\delta_{ext}(x) = \frac{\#external\_edges}{n_x(n - n_x)}$$

Fórmula 3: Cálculo de separabilidad

Donde:

  - #**external_edges**: es el número de conexiones que tienen los nodos de la comunidad con nodos de otras comunidades.
  - $\boldsymbol{n_x}$: corresponde al número de nodos de la comunidad x sobre la que se está aplicando la fórmula de cohesión.
  - **n**: es el número total de nodos del grafo.

La idea de esta definición de separabilidad es premiar a las comunidades que tengan pocas conexiones con otras comunidades frente a otras que tengan muchas conexiones.

Esta función tiene un rango de valores entre [0,1], siendo el valor 1 el peor caso, en el que el nodo estuviese en una comunidad y tuviese una conexión con el resto de los nodos que estuviesen en otras comunidades. El máximo valor equivaldría a tener a todos los nodos de una comunidad sin conexiones con otros nodos de otras comunidades.

Por lo tanto, definimos el fitness individual para una comunidad x como la resta de su cohesión y su separabilidad, buscando de esta manera una alta cohesión y una baja separabilidad. Es posible que para soluciones poco óptimas obtengamos un fitness negativo.

$$fitness_X = \delta_{int}(x) - \delta_{ext}(x)$$

Fórmula 4: Fitness individual

Pero lo más probable, y lo que se espera, es que nuestro algoritmo de detección de comunidades no nos clasifique todos los usuarios de nuestro grafo en una sola comunidad, sino que decida realizar una partición de diferentes comunidades. Por lo tanto debemos aplicar la fórmula del fitness individual para cada comunidad y hacer la media entre el número de comunidades a las que se aplica la función de fitness, es decir, las comunidades que ha clasificado el algoritmo.

$$fitness = \left(\sum_{\forall c} \frac{\delta_{int}(x) - \delta_{ext}(x)}{\#comunidades} \right)$$

Fórmula 5: Fitness general

Este sería el fitness general del grafo para la solución creada por una hormiga tras el recorrido del grafo y la división de los nodos en comunidades. Por lo tanto, este será el valor que guardaremos en nuestra estructura de soluciones junto con la solución de las comunidades detectadas para posteriormente elegir entre ellas la mejor, que se corresponderá con la que tenga un valor de fitness más alto.

3.6 Selección de comunidad

Para poder resolver el problema de detección de comunidades en una red social, es evidente que los nodos de la red tendrán que pasar por un proceso de asignación de comunidad.

En este Trabajo de Fin de Grado las encargadas de realizar el proceso de selección de comunidad de un nodo son las hormigas cuando visitan el nodo que se quiere asignar a una comunidad.

Las hormigas se mueven por el grafo de nodo en nodo de forma aleatoria, esto hace que finalmente las hormigas recorran todos y cada uno de los nodos asignando así una comunidad a todos ellos. Para poder realizar correctamente la selección de comunidades se necesita cierta información para hacer la toma de decisión.

Cuando una hormiga pasa por un nodo, recoge la información que este contiene: las feromonas que dejaron las hormigas anteriores cuando clasificaron ese nodo en sus soluciones. En ese momento se obtiene también la heurística de la red sobre ese nodo, y, junto con los valores que porta la hormiga α y β, las cuales darán más importancia a la heurística o a las feromonas respectivamente, se realizará la toma de decisión.

Se describe la probabilidad de que una hormiga k inserte el nodo i en la comunidad j como:

$$p_{ij}^{k} = \frac{\tau_{ij}^{\alpha} * \eta_{ij}^{\beta}}{\sum_{u \in \mathcal{N}_i^k} \left(\tau_{iu}^{\alpha} * \eta_{iu}^{\beta}\right)}$$

Fórmula 6: Probabilidad selección de comunidad

Donde:

- $\boldsymbol{\tau_{ij}^{\alpha}}$: corresponde con el valor calculado de la heurística del nodo i y la comunidad j.
- $\boldsymbol{\eta_{ij}^{\beta}}$: corresponde al valor de las feromonas del nodo que tengan relación con los nodos de la comunidad j
- $\boldsymbol{\alpha}$**:** valor que elevará al resultado de la heurística.
- $\boldsymbol{\beta}$: valor que elevará el resultado de las feromonas coincidentes.
- $\boldsymbol{\tau_{iu}^{\alpha}}$**:** valor de la heurística con una comunidad u (serán todas las existentes al recorrer el sumatorio)
- $\boldsymbol{\eta_{ij}^{\beta}}$: valor de las feromonas coincidentes con una comunidad u (serán todas las existentes al recorrer el sumatorio)

4 Desarrollo

Con el fin de conseguir nuestro objetivo en este Trabajo de Fin de Grado de desarrollar un algoritmo que de una solución al problema de la detección de comunidades en las redes sociales basado en el trabajo de optimización de la colonia de hormigas (ACO), se han utilizado diferentes fuentes de datos y herramientas que se describirán a continuación.

De la misma manera en esta sección se explicará el proceso de desarrollo del algoritmo según el diseño creado. **Ver apartado 3.2** Se desarrollarán con detalle las funciones creadas para realizar todas las tareas necesarias para el correcto funcionamiento del algoritmo, al igual que la estructura de ficheros elegida y se pondrá un ejemplo al final para ilustrar y hacer más sencilla la comprensión para el lector del proceso que sigue una hormiga a lo largo del grafo.

4.1 Lenguaje de programación

Para el desarrollo del algoritmo propuesto hemos utilizado el lenguaje de programación Python.

Python es un lenguaje de programación que creó el holandés Guido Van Rossum y que nació de la necesidad de un lenguaje de orientación a objetos que fuese sencillo y que cubriese ciertas tareas de programación que en ese momento se estaban haciendo en Unix con C.

Hemos elegido este lenguaje de programación por la amplia cantidad de ventajas que presentaba frente a otros lenguajes de programación.

- Es un lenguaje fácil de aprender y que nos ofrece una gran variedad de estructuras de almacenamiento de datos como diccionarios, listas, tuplas y lo más importante, es muy fácil tratar y operar con todas ellas.
- Python tiene la ventaja de que existen una amplia variedad de paquetes y librerías externas que sirven de apoyo en muchos problemas. En particular para este proyecto hemos utilizado una librería matemática, una librería para crear grafos y otra para pintar dichos grafos. La más importante de todas es la librería NetworkX encargada de crear y manejar el grafo que recorrerán las hormigas.
- Es un lenguaje que está orientado a ser sencillo y eficiente por lo que trabajar con él facilita mucho las tareas de desarrollo. También se caracteriza por ser un leguaje legible por lo que será agradable a la vista de otros usuarios que deseen entender el código desarrollado.
- Es un lenguaje muy utilizado en el desarrollo de este tipo de problemas biométricos por lo que la mayoría de los trabajos previos relacionados con el nuestro están desarrollados en Python.

- Otra de las ventajas de Python es que no tiene problema en tratar con grandes volúmenes de datos que es precisamente una de las limitaciones que podríamos encontrar en algún otro lenguaje de programación para la resolución de este problema que realiza un estudio sobre un subgrafo de la red de Facebook.

4.2 Dataset de Facebook

La red social sobre la que vamos a resolver el problema de detección de comunidades es una red real y es una parte de la red social Facebook. Estos datos los hemos obtenido de un estudio de la Universidad de Stanford.

Los datos fueron recogidos con el permiso de sus usuarios a través de la aplicación Facebook App. Este dataset de datos está provisto de información de los perfiles de los usuarios, comunidades existentes entre ellos en base a sus perfiles y está dividido en redes ego.

Es muy importante mencionar que toda la información que se puede obtener de este estudio y que está al alcance de cualquier usuario de la red ha sido previamente anonimizada reemplazando todos los ids internos de Facebook con otros ids. La interpretación de las características de los perfiles ha sido renombrada con otras características diferentes.

Existen también datasets de las redes sociales Google+ y Twitter, pero finalmente se ha elegido la red social Facebook por los siguientes motivos:

- Facebook es una de las redes más importantes y famosas del mundo, es también más antigua que Google+ y Twitter por lo que es probable que más usuarios hayan utilizado Facebook alguna vez y por lo tanto se sientan en parte identificados.
- Las conexiones de Facebook se interpretan con la "amistad". Esto quiere decir que dos nodos del grafo estarán unidos (dos usuarios estarán unidos) si entre ellos comparten una relación de amistad. Podemos traducir esto a la teoría de grafos como un grafo bidireccional **Ver Figura 1.** Este hecho facilita mucho más el algoritmo que si fuese un grafo dirigido como sería el caso de Twitter en el que las relaciones entre los usuarios están dadas por "seguimiento". Un usuario puede seguir las publicaciones que realiza otro usuario, pero esto no implica que este otro usuario deba seguir al primero, al no cumplir una condición de reciprocidad lo traduciríamos en la teoría de grafos a un grafo no dirigido **Ver Figura 2** que sería más difícil de analizar.
- Google+ tampoco cumple al igual que Twitter la condición de reciprocidad ya que las relaciones también son de "seguimiento" por lo que tampoco nos aplica para este algoritmo. Otro motivo por el que se ha elegido Facebook por encima

de Google+ o Twitter es porque el volumen de datos del dataset de Facebook es mayor.

El dataset se compone de 10 redes ego, que corresponden a 10 usuarios diferentes de la red en la que como explicábamos previamente los datos tanto de los perfiles como de las conexiones han sido anonimizados.

Cada una de estas 10 redes ego se compone de 5 ficheros diferentes que describen la red del usuario en cuestión. Estos ficheros poseen una nomenclatura en particular para diferenciar sus propósitos y el usuario al que hacen referencia. Siguen la siguiente forma: [id nodo].[tipo fichero] Los 5 tipos diferentes de fichero se describen a continuación:

- **nodeId.edges**: en este fichero se encuentran las conexiones de los usuarios de la red para el nodo 'nodeId'. Las conexiones son no dirigidas para este dataset de Facebook, en el caso de Twitter o Google + serían dirigidas, lo que significa que en este fichero las entradas implican reciprocidad. El fichero contiene entradas con los ids de los nodos de la siguiente manera "1 2" que significa que el nodo 1 estará conectado con el nodo 2. El nodo ego que en este caso sería nuestro nodeId no aparece en este fichero pues se asume que al ser una red ego la que se está tratando todos los nodos que aparezcan en este fichero (alters) estarán conectados al nodeId (ego)
- **nodeId.circles**: fichero en el que se representan las comunidades (círculos) formadas por los perfiles de los usuarios. Cada línea corresponde a una comunidad donde la primera entrada de cada línea es el nombre de la primera comunidad (circulo)
- **nodeId.feat:** este fichero contiene todas las características de los nodos que aparecían en el fichero .edges. Están representadas con ceros y unos, es decir en binario). El valor 0 corresponde a no cumplir la característica y el valor 1 corresponde a sí cumplirla. Estas características están definidas en el fichero .featnames. El fichero se compone de una línea por nodo y sus características separadas por espacios, por ejemplo: "1 0 0 1 0 1 0 1"
- **nodeId.egofeat:** aquí al igual que en el fichero .feat se guardan características solo que en este caso en particular la información almacenada son las características del nodo nodeId (ego)
- **nodeId:featnames:** en este fichero es donde se guarda la correspondencia de los ceros y unos definidos en los ficheros .feat y .featego. Es decir, guarda los nombres de las características que existen en la red. La información de este fichero también ha sido anonimizada ya que los nombres reales de las características desvelarían información real sobre la privacidad de los usuarios.

El fichero .feat sería un fichero muy importante en el caso de que se hubiese decidido realizar un análisis de detección de comunidades orientado a los perfiles de los usuarios, es decir a sus características. En este caso al estar nuestro algoritmo enfocado al análisis

del problema de comunidades mediante la topología de la propia red, o lo que es lo mismo, por sus conexiones, hace que el fichero .edges cobre más importancia.

El fichero más importante de este trabajo es sin duda alguna el fichero .edges que es el fichero que pasaremos a la función de creación del grafo de la herramienta NetworkX que iterará sobre este fichero línea a línea creando los nodos del grafo y sus conexiones. Podemos encontrar este proceso en el fichero g*rafo.py*.

4.3 Estructura de ficheros

Para la fácil implementación del algoritmo de detección de comunidades se ha seguido una estructura de ficheros para su correcta organización, sencillez a la hora de programar y por supuesto para mantener cierta coherencia en el código. En este apartado se expondrá una breve descripción de cada uno de los ficheros que forman el proyecto.

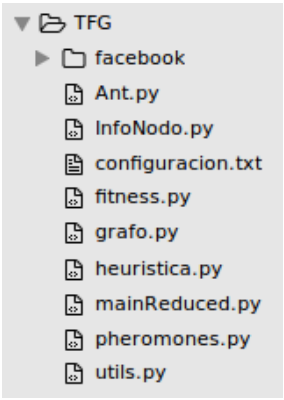


Figura 10: Estructura del proyecto

- **Carpeta Facebook**: contiene los ficheros de las 10 redes ego del dataset de Facebook descargado del estudio de la Universidad de Stanford explicados en el punto 4.2
- **Main.py**: este es el fichero principal del algoritmo, donde empieza y donde se encuentran los bucles por N.º Repeticiones, N.º Iteraciones y hormigas. Desde aquí se crea cada hormiga que se mandará a recorrer el grafo, se llamará a las funciones de cálculo de fitness y almacenará las soluciones locales de las hormigas, las iteraciones y repeticiones. Finalmente, desde aquí se llama a la función que obtiene la mejor de todas estas soluciones guardadas.
- **Configuración.txt**: archivo donde se guardan los parámetros de configuración del algoritmo: N.º Repeticiones, N.º Iteraciones, N.º de hormigas, alfa, beta y el ratio de evaporación de las feromonas. **Ver apartado 3.2**.
- **Grafo.py**: en este fichero encontramos la funcionalidad referente al grafo. Aquí se incluyen las siguientes funciones:
    - `readGraph(nombreArchivo):` encargado de leer del archivo .circles que se pasa por parámetro del usuario ego del que queramos hacer el análisis y

crear el grafo utilizando la librería NetworkX con la función `read_edgelist`. En esta función se añadirán como comentábamos en capítulos anteriores el nodo ego que no está en el archivo .circles y una conexión con todos sus alters (el resto de los nodos del grafo)

  - `initializeNodeAttributes(G, nombreAtributo):` función encargada de inicializar el estado visitado de los nodos del grafo para que las hormigas puedan circular por él.
  - `cargarConfiguracion(archivo):` método que parsea el archivo de configuración y guarda los parámetros con los que se ejecutará el algoritmo sobre el grafo.

- **InfoNodo.py**: fichero en el que se define qué información contendrán los nodos del grafo por los que pasarán las hormigas. Esta información es el estado de visitado y el diccionario de feromonas. **Ver apartado 3.2**.
- **Ant.py**: este fichero es uno de los más importantes ya que describe cómo está formada la hormiga y la clave del algoritmo que es el comportamiento de la hormiga a través del grafo. En este fichero encontramos las siguientes funciones principales:
  - `recorreGrafo(self, G, nodoInicial) :` la cual como bien su nombre indica, es la encargada de mover a la hormiga por el grafo y realizar sus acciones. Se detallará con más profundidad en el **apartado 4.4**
  - `selectComunidad(self, normalizado, pherDict):` en esta función se realizará la selección de comunidad por parte de la hormiga. Se detallará con más profundidad en el **apartado 4.7**
  - `calculaPheromonesComunidad(self, comunidad, pherDict):` esta función se encarga de calcular cómo afectan las feromonas existentes en el nodo a la toma de decisión sobre la asignación de la comunidad en la solución actual. Se profundizará más en esta función en el **apartado 4.6**
- **Fitness.py**: archivo en el que se almacenan las funciones relacionadas con el cálculo del fitness de la solución local. Se detallará más adelante en el **apartado 4.8**
- **Heuristica.py**: fichero que contendrá las funciones relacionadas con el cálculo de la heurística orientada a la topología de la red. Se detallará más adelante en el **apartado 4.5**
- **Pheromones.py**: en este fichero se incluyen las funciones que tratan directamente con las feromonas como obtener feromonas de un nodo, El proceso de depositar, evaporar, o limpiar las feromonas de grafo. Estas funciones se verán con más detalle en el **apartado 4.6.**
- **Utils.py**: en este archivo se encuentran funciones auxiliares para el desempeño del resto de funciones principales.

4.4 Recorrido de la hormiga por el grafo

Las hormigas son las estructuras principales de este algoritmo de detección de comunidades y se espera obtener una respuesta a este problema para nuestro grafo a en su comportamiento, pero para poder determinar las comunidades de la red, la hormiga debe de ser capaz de recorrer todos los nodos de esta para poder realizar correctamente la tarea de partición en comunidades. Puesto que todos los nodos deben de ser asignados a una comunidad para poder tener una solución válida, aunque haya nodos que formen su propia comunidad, no significa que la hormiga no los ha visitado sino que la hormiga los visitó y se decidió que ese debía de ser su estado.

En nuestro algoritmo se ha definido una función que represente el movimiento de la hormiga por el algoritmo y las acciones que realiza en su proceso. Esta función es:

```
recorreGrafo(self, G, nodoInicial)
```

Esta función propia de la hormiga es la encargada de mover a la hormiga por el grafo G y nodoInicial como bien se puede intuir, será el nodo desde donde empiece a recorrerlo (en nuestro desarrollo lo hacemos empezar en el nodo 'ego')

Para recorrer el grafo se calcula en el momento inicial los vecinos del nodo desde el que se desea empezar, como en nuestro caso el nodo inicial es el nodo 'ego', los vecinos de este nodo serán por definición el resto de los nodos de la red. Por lo tanto, tendremos una lista con todos los nodos que querremos tratar en el problema de detección de comunidades ya que el nodo 'ego' no entra dentro de la división como se explicaba en el apartado de diseño.

Nuestro objetivo entonces será recorrer los nodos del grafo y e ir asignándolos una comunidad, y a medida que estos nodos son visitados se cambia su estado a visitado y se eliminan de la lista creada. El bucle de recorrido finalizará cuando no queden nodos en esta lista, lo que significará que el algoritmo los ha recorrido todos.

Dentro del bucle se elegirá el próximo nodo que visitar de forma completamente aleatoria de entre los vecinos del nodo actual.

```
seleccionarProximoNodo(vecinos_sin_visitar):
```

En el caso de la primera iteración donde el nodo actual es el 'ego', los vecinos_sin_visitar serán por tanto todos los nodos del grafo pudiendo elegirse cualquiera. Pero más adelante si el nodo 5 tiene como vecinos sin visitar todavía a los nodos 2, 3, 4 y 'ego' solo podrá elegir entre esos cuatro nodos para moverse. El nodo 'ego' tiene una condición de visitado especial, al no clasificarse siempre puede ser accedido por muchas veces que el algoritmo lo seleccione como próximo nodo. Esto es crucial porque es lo que permite que se puedan recorrer todos los nodos del grafo ya que el nodo 'ego' siempre está disponible para la selección de próximo nodo y cuando no haya más vecinos sin visitar más que él (recordamos que siempre aparecerá la posibilidad de moverse al nodo 'ego') la probabilidad de moverse a este será del 100%.

Por lo tanto, gracias a esto y que por la propia descripción de red ego el nodo 'ego' está conectado con el resto de los nodos, nunca quedarán nodos sin visitar.

Cuando el nodo 'ego' sea elegido simplemente se saltarán todas las operaciones relativas al análisis de la red y selección de comunidad y se seleccionará de nuevo otro nodo, pero esta vez el nodo actual será el nodo 'ego' por lo que todos los nodos del grafo que no hayan sido visitados serán sus vecinos.

Cuando se selecciona un nodo que no sea el nodo 'ego, cuyo tratamiento ya hemos comentado, comenzamos el análisis de este calculando la heurística de este nodo en las comunidades actuales y posteriormente normalizando el resultado para incluir la posibilidad de añadir una nueva comunidad **Ver apartado 4.5**. Más adelante, se obtienen las feromonas depositadas por hormigas anteriores en el nodo actual **Ver apartado 4.6** y se procede a la selección de comunidad gracias a toda la información que se acaba de recoger **Ver apartado 4.7**. A continuación, cuando una comunidad ha sido elegida para el nodo, actualizamos la solución local de la hormiga.

```
updateLocalSolution(nodo_actual, selectedCluster)
```

Finalmente, tras haber analizado y adjudicado una comunidad al nodo, se debe marcar como visitado para que no pueda volver a ser elegido en el proceso de selección de próximo nodo y se elimina de la lista inicial de nodos que el grafo debe de recorrer antes de terminar el proceso completo de detección de comunidades.

4.5 Proceso de cálculo de heurística y normalización

Uno de los dos pilares en la toma de decisión de la asignación de comunidades es la función heurística. Para este algoritmo se ha utilizado una heurística que toma en consideración la información obtenida de la topología de la red, es decir, de las conexiones ente los nodos.

El proceso de recoger esta información es llevado a cabo cuando la hormiga visita un nodo, porque recordemos que la heurística es una de las partes a tener en cuenta para asignar la comunidad al nodo que la hormiga está visitando. Por lo tanto, este proceso se realizará una vez por cada nodo del grafo diferente del nodo 'ego' que no se clasifica ya que todos los demás deben de tener una comunidad asociada.

La hormiga calcula la heurística con todas las comunidades actuales de su solución local mediante la función:

```
getEvaluatedClusters(G, solucionLocal, nodo_actual):
```

Esta función calcula los vecinos del nodo actual y para cada comunidad de la solución local aplica la Fórmula 1 y devuelve un diccionario de clave la comunidad con valor la heurística para esa comunidad.

Este proceso evalúa la heurística de todas las comunidades existentes en la solución local y si no se hiciese nada más, la hormiga solo podría seleccionar comunidad entre las existentes pudiendo llevar a soluciones poco óptimas. Esto crea un problema y es el de cómo y cuándo crear una nueva comunidad.

```
normalizacion(G, heuristicDict, nodo_actual)
```

El problema de la creación de una probabilidad se soluciona normalizando estadísticamente el diccionario de heurísticas de las comunidades. Esto significa que debemos ajustar los valores medidos de las heurísticas de las comunidades a una escala común que será del [0-1] que indicará la probabilidad de ser elegida en función de sus valores de heurística mientras añadimos una nueva comunidad que se calcula de la siguiente manera:

$$p_{Cnueva} = \frac{1}{\#Vecinos\ nodo\ actual \operatorname{sin} ego}$$

Fórmula 7: Probabilidad de nueva comunidad

Por lo tanto, el diccionario normalizado que se crea contiene como clave las comunidades incluyendo la nueva comunidad creada con la fórmula anterior y como valor la probabilidad de ser elegidas [0-1]

4.6 Tratamiento de feromonas

El otro pilar a la hora de tomar la decisión de selección de comunidad son las feromonas que dejan las hormigas pasadas sobre los nodos. El hecho de que las hormigas depositen las feromonas sobre los nodos en el caso de nuestro algoritmo o en las conexiones o de cualquier otra forma es lo que hace posible que puedan interactuar entre ellas y esa es la clave de las técnicas basadas en el algoritmo de la colonia de hormigas (ACO)

Cuando una hormiga visita un nodo que no sea el 'ego', ya que como se ha explicado anteriormente este nodo no se clasifica y por lo tanto tampoco tiene feromonas de soluciones anteriores, debe recoger la información de las feromonas alojadas en dicho nodo y posteriormente utilizar estas feromonas en el proceso de selección de comunidad.

Para tratar la información de las feromonas se han creado diferentes funciones. Una función de obtención de las feromonas, otra para depositarlas en los nodos, otra que aplica la evaporación de las feromonas del grafo y finalmente una que limpie las feromonas del grafo.

- `getPheromoneInformation(G, nodo_actual):` método encargado de recoger la información de las feromonas del nodo cuando la hormiga llega a dicho nodo. Se realiza un acceso al campo del InfoNodo que contiene el diccionario de feromonas.

- `depositPheromones(G, solucionLocal, fitness):` método encargado de depositar las feromonas en los nodos al final del recorrido de la hormiga por el grafo. La información que se guarda es el fitness de la solución obtenida y tiene la siguiente estructura:

Relación de nodos	Fitness
(3,4)	1,78
(3,5)	1,5
(3,)	0,56

Tabla 1: Ejemplo de diccionario de feromonas

Esta sería una hipotética situación en la que una hormiga k llega al nodo 3 que contiene este diccionario de feromonas. La primera entrada en este diccionario indica que hormigas pasadas unieron el nodo 3 y el nodo 4 en una o más ocasiones en la misma comunidad y la columna de fitness corresponde a la suma de los fitness de la resolución del problema de detección de comunidades de dichas hormigas.

El caso (3,) es la manera de simbolizar que una o varias hormigas evaluaron el nodo 3 en una comunidad independiente donde este nodo era el único que formaba dicha comunidad.

- `evaporatePheromones(G, ratio)`: esta función se encarga de aplicar el parámetro de evaporación de feromonas cuando finaliza una iteración. Accede a todos los nodos del grafo y va uno por uno multiplicando el ratio de evaporación por los fitness de cada entrada de la **Tabla 1** de los nodos. Recordamos que la importancia de que las feromonas se vayan perdiendo a lo largo del tiempo es muy grande ya que esto nos permite no estancarnos en soluciones máximas locales.
- `clearPheromones(G):` método encargado de limpiar los diccionarios de feromonas guardados en los nodos cuando finaliza una repetición. Recordamos que cada repetición del algoritmo es independiente de los resultados de las repeticiones anteriores por lo que es obligatorio hacer una limpieza de feromonas para dejar el grafo virgen para la siguiente repetición.

4.7 Proceso de selección de comunidad

Para resolver nuestro problema, el algoritmo debe contener una fase de selección de comunidad para cada nodo del grafo que no sea el nodo 'ego'. Este proceso lo realiza la hormiga cuando visita un nodo en base a la información recogida tanto de la función heurística del problema como de las feromonas depositadas por las hormigas pasadas.

```
selectComunidad(self, normalizado, pherDict)
```

Recordamos que para la fórmula de probabilidad de que el nodo i pertenezca a la comunidad j se multiplicaba el cálculo de la heurística de esa comunidad elevado a su grado de relevancia α por el cálculo de las feromonas que afectan a la solución local de la comunidad j desde el nodo i elevado a su grado de relevancia β. Todo ello se divide por la suma de estos cálculos para todas las comunidades como se observa en la **Fórmula 6.**

Por lo tanto, lo primero que haremos será montar el denominador de la fórmula que contiene el sumatorio. Por cada comunidad del diccionario normalizado que contiene las probabilidades de las comunidades incluyendo la posibilidad de incorporar una nueva comunidad. Se multiplica el valor de cada una de estas probabilidades normalizadas por el cálculo de feromonas guardadas en el nodo que afectan a esa comunidad.

El cálculo de estas feromonas se realiza en la función:

```
calculaPheromonesComunidad(self, comunidad, pherDict)
```

Donde se obtiene la lista de los nodos pertenecientes a la solución local de la hormiga de esa comunidad y se cruza con los nodos relacionados anteriormente en el diccionario de feromonas como se simbolizaba en la **Tabla 2**. Por cada nodo de la comunidad del que se encuentre entrada en el diccionario de feromonas se acumula su fitness y finalmente se devuelve la suma total del fitness de todos los nodos de la comunidad que estaban anteriormente evaluados en la misma comunidad por otras hormigas.

En el caso de que no se tengan nodos en la comunidad elegida, existe la posibilidad de incorporar una nueva comunidad por lo que se buscará en el diccionario de comunidades la entrada (nodoActual,) que era el supuesto en el que otra hormiga evaluó en el pasado al nodo actual en una comunidad donde él era el único nodo. Si existiese dicha entrada recogerá su valor fitness.

Si no se encuentran feromonas para una comunidad, tanto en el caso de búsqueda en las comunidades existentes como en el caso de nueva comunidad, se devolverá 1.0 como valor de las feromonas para no hacer 0 el numerador y denominador de la ecuación.

El proceso de selección de la comunidad se realizará sacando un número aleatorio en el rango [0,1] por lo que tenemos que adecuar las probabilidades de selección de cada comunidad.

Por este motivo creamos un diccionario de probabilidades con el cálculo de cada función heurística y las feromonas para cada comunidad dividido entre el total recién calculado.

```
creaDictProb(probabilidades)
```

En este diccionario se formarán los rangos de probabilidad de elección de las comunidades en función de los cálculos realizados previamente, obteniendo un diccionario como el siguiente:

Comunidad	Rango de probabilidad
0	[0.0, 0.35]
1	[0.35, 0.78]
2	[0.78, 0.78]
3	[0.78 , 1.0]

Tabla 2: Ejemplo de rangos de probabilidad de elección de comunidad

Finalmente se llama a la función creada para determinar aleatoriamente la comunidad:

```
selectRandomCom(self, dictProbabilidades)
```

Donde se obtendrá un número aleatorio entre [0,1] y se buscará en qué comunidad cae. Esto implica que, aunque una comunidad tenga muchas posibilidades de ser elegida, no siempre tendrá por qué ser esa la solución. Esto tiene consecuencias positivas y negativas, ya que evita caer en soluciones poco óptimas que han sido elegidas por las primeras hormigas por lo que elimina poder a las primeras hormigas y no hace el algoritmo tan dependiente de estas. Por otro lado, puede desviarse de una buena solución, sin embargo, si la solución es buena seguirá teniendo buenas posibilidades de ser seleccionada por futuras hormigas frente a otras que no tengan una solución tan adecuada. Para esto influyen los valores de α y β los cuales dan más prioridad a la heurística o a las feromonas.

4.8 Cálculo de fitness de una solución

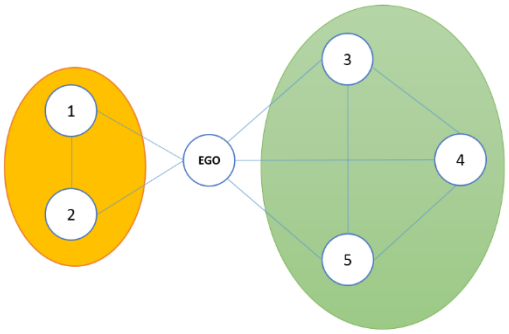
Para poder determinar cómo de buena es la solución propuesta por una hormiga para el problema de detección de comunidades, el algoritmo se basa en una función de fitness definida en el **Apartado 3.5** que recordemos que premiaba las comunidades con muchas conexiones entre sus nodos y penalizaba comunidades que tuviesen una gran cantidad de conexiones ente los nodos de su comunidad y nodos de otras comunidades.

El momento en el que la hormiga finaliza de recorrer el grafo y ha visitado y adjudicado una comunidad a todos los nodos de este, a excepción del nodo ‘ego’, es indicativo de que la hormiga ha terminado su proceso y por lo tanto tiene construida una solución para el problema.

El paso lógico por lo tanto es evaluar la calidad de esta solución, este procedimiento se realiza en la función:

```
calcula_fitness(G, diccionario_comunidades)
```

En ella se evalúa la solución en función de la cohesión y la separabilidad explicados en el Apartado 3.5. Partiendo de una solución, por ejemplo:



Comunidad	Nodos comunidad
0	[1,2]
1	[3,4,5]

Tabla 3: Ejemplo de solución de una hormiga

Figura 11: Ejemplo de solución de una hormiga

La función evalúa la cohesión y separabilidad para cada comunidad mediante las funciones:

```
cohesion(G, comunidad)
```

```
separabilidad(G, comunidad, comunidades)
```

Posteriormente, se obtienen los fitness individuales para cada comunidad mediante la **Fórmula 4** y se engloba cada uno de ellos mediante la **Fórmula 5** obteniendo así el fitness general de la solución. Finalmente, esta solución que ha evaluado la hormiga se guardará para elegir, en último lugar cual es la mejor solución de todas las hormigas tras las repeticiones e iteraciones parametrizadas.

4.9 Estructura de soluciones y obtención de la mejor solución

Cuando una hormiga finaliza su proceso de asignación de comunidades y evalúa la solución obtenida del problema la guardamos para tener un registro al cual acceder al final del algoritmo. Este registro se va actualizando de la siguiente manera:

- Un diccionario con las soluciones de todas las hormigas definidas en el fichero de configuración como N. º de hormigas.

Fitness	Solución
12,55	0 : [1,2] 1 : [3,4,5]
7,20	0 : [1,2] 1 : [3] 2 : [4,5]
2,15	0 : [1] 1 : [2] 2 : [3,4] 3 : [5]

Tabla 4: Ejemplo de estructura de soluciones

Cuando la solución evaluada tiene el mismo fitness que una de las entradas del diccionario de soluciones, se sobrescribe en su posición para mantener una estructura sencilla y organizada. La inmensa mayoría de las veces se tratará de la misma descomposición de comunidades.

Como la red bajo estudio tiene un volumen de datos tan grande podemos descartar casi con toda probabilidad que se dé el caso en el que exista una partición de comunidades con el mismo fitness que otra partición diferente de comunidades. Si existiese tampoco sería un problema porque la información perdida tendría exactamente la misma calidad que la solución nueva que la está sobrescribiendo.

Cuando finalizan las N hormigas de evaluar sus soluciones y rellenar el diccionario, se selecciona la mejor solución de una de las hormigas del conjunto de hormigas de la iteración y se guarda en un nuevo diccionario de mejores soluciones del conjunto de hormigas de una iteración. Todos los procesos de seleccionar la solución con mayor fitness de los diccionarios de soluciones se realizan mediante la función:

```
maxFitnessSolucion(self, soluciones)
```

Fitness	Solución
12,55	0 : [1,2] 1 : [3,4,5]
7,20	0 : [1,2] 1 : [3] 2 : [4,5]
2,15	0 : [1] 1 : [2] 2 : [3,4] 3 : [5]

Tabla 5: Ejemplo de estructura de soluciones de las iteraciones

Fitness	Solución
12,55	0 : [1,2] 1 : [3,4,5]

Tabla 6: Ejemplo de estructura de soluciones de las repeticiones

5 Integración de pruebas y resultados

Si se hubiese seguido una estructura de trabajo donde se pasa a la fase experimental una vez finalizadas las fases de análisis y desarrollo del algoritmo, no hubiésemos sido capaces de encontrar algunos errores en el diseño y posterior implementación del algoritmo.

Las pruebas se han ido realizando sobre las diferentes secciones del algoritmo una vez se finalizaban para probar su correcto funcionamiento y así no esperar al último momento donde muy probablemente se obtuviesen resultados diferentes a los conseguidos y encontrar el error que provoca estos comportamientos resultase más complejo.

Por lo tanto, se pueden diferenciar entre dos tipos de pruebas: pruebas unitarias sobre los diferentes componentes del algoritmo y pruebas generales que engloban todas las secciones del algoritmo y de las que obtendremos los resultados sobre los que analizaremos las soluciones del problema de detección de comunidades.

5.1 Pruebas locales

Para desarrollar el algoritmo se ha utilizado un grafo de prueba más pequeño donde se pudiese observar el comportamiento asociado a la detección de comunidades a simple vista para poder comprobar los resultados obtenidos de una manera más rápida y eficaz.

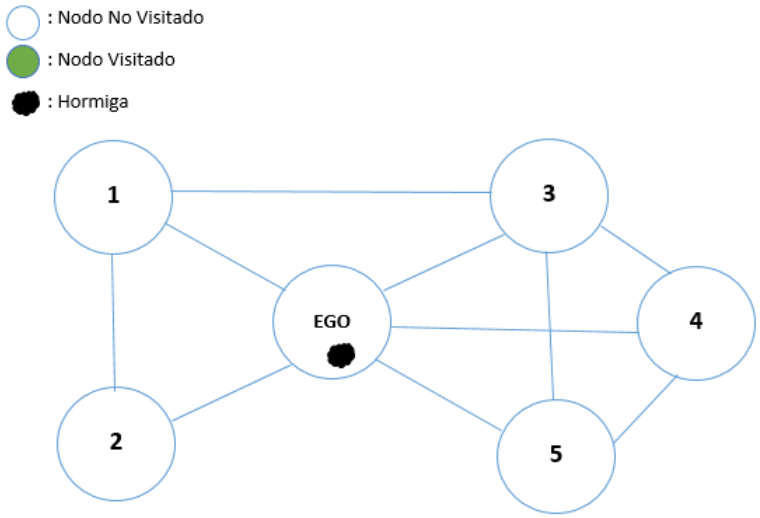


Figura 12: Red de prueba sencilla

La hormiga comienza en el nodo ‘ego’ y decide aleatoriamente qué nodo visitar entre sus vecinos no visitados, en este caso puede elegir entre todos los nodos. Podemos observar a simple vista como los nodos 1 y 2 formarán una comunidad mientras que los nodos 3, 4 y 5 formarán otra. Recordamos que, para cada nodo, se debe realizar el proceso de selección de comunidad y para ello es necesario calcular la heurística y las

feromonas. Posteriormente marcamos como visitado el nodo y avanzamos al siguiente nodo sin visitar de la misma manera donde se realizarán los mismos cálculos. Ejecutamos el algoritmo con los siguientes valores:

Parámetro	Valor
Repeticiones	10
Iteraciones	25
Hormigas	25
α	3.0
β	1.0

Tabla 7: Parámetros de entrada prueba sencilla

Finalmente, cuando la hormiga ha terminado de recorrer la red y ha realizado la partición en comunidades correspondiente, es el momento de evaluar el fitness de la solución mediante la aplicación de los factores de cohesión y separabilidad a cada comunidad como se explicaba en el **apartado 3.5.**

El resultado de esta solución sería el siguiente:

Fitness	0,833
Comunidades	0 : [3, 4, 5] 1 : [1, 2]
Tiempo (s)	9,578

Tabla 8: Solución prueba grafo sencillo

Este resultado coincide con el resultado que pronosticábamos con anterioridad a simple vista gracias a que es un grafo pequeño y sencillo. El resultado obtenido es de 0,833 el cual es un buen resultado si contamos que el mínimo es 0 y el máximo es 1. El tiempo empleado en la ejecución es un tiempo no muy elevado. Por lo tanto, el algoritmo se comporta dentro de la lógica planteada para nuestro caso de prueba con un grafo sencillo ya que detecta las comunidades esperadas de la red con un fitness bueno y adjudica las mismas comunidades que habíamos detectado nosotros con un simple vistazo al grafo antes de iniciar cualquier operación.

Una vez comprobado que el algoritmo se comporta tal y como se esperaba para un grafo sencillo, se procede a hacer las pruebas con el dataset de Facebook proporcionado por la Universidad de Stanford descrito en el **apartado 4.2**

Este dataset contiene 10 redes ego, nosotros haremos pruebas sobre 3 de ellas ya que como veremos más adelante, las pruebas con redes grandes son bastante costosas.

Ejecutaremos el algoritmo sobre estas redes con distintos parámetros de entrada buscando una relación entre estos y el fitness calculado por el algoritmo. Los parámetros

que modificaremos en estas pruebas serán las iteraciones y las hormigas dejando estáticos el resto de los parámetros de entrada.

5.2 Resultados dataset de Facebook

Finalmente se ha decidido hacer las pruebas sobre las redes de los usuarios 3980, 698 y 414 ya que eran las redes con menor cantidad de nodos y por lo tanto en las que el algoritmo tardará menos en ejecutar las pruebas.

Red Ego (Usuario)	Tamaño Red (Nodos)
0	334
107	1035
1684	787
1912	748
3437	535
348	225
3980	53
414	151
686	169
698	62

Tabla 9: Tamaño de las redes dataset Facebook

Se han realizado diferentes tipos de pruebas para cada red:

- 50 iteraciones y hormigas variables (α y β intercambiados para nodo 3980)

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	3	1	0,7171	37,472
10	50	25	3	1	0,7222	94,459
10	50	50	3	1	0,7286	187,55
10	50	75	3	1	0,7624	281,93
10	50	100	3	1	0,7575	372,72

Tabla 10: Pruebas usuario 3980 (1)

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	1	3	1,5315	38,52
10	50	25	1	3	1,5598	95,9
10	50	50	1	3	1,5614	189,14
10	50	75	1	3	1,5956	284,74
10	50	100	1	3	1.5871	369,15

Tabla 10.5: Pruebas usuario 3980 (1.5)

- 25 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	25	3	1	0,668	18,67
10	25	25	3	1	0,7002	47,56
10	50	25	3	1	0,7228	93,59
10	75	25	3	1	0,74	143,52
10	100	25	3	1	0,80511	187,7
10	125	25	3	1	0,7471	236,73
10	150	25	3	1	0,7939	281,93
10	175	25	3	1	0,777	328,39
10	200	25	3	1	0,764	377,95

Tabla 11: Pruebas usuario 3980 (2)

- 50 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	50	3	1	0,6924	37,4
10	25	50	3	1	0,7083	94,086
10	50	50	3	1	0,7583	186,41
10	75	50	3	1	0,716	284,28
10	100	50	3	1	0,72	368,59

Tabla 12: Pruebas usuario 3980 (3)

El resto de las pruebas realizadas se encuentran en el **Anexo A**.

5.3 Discusión sobre los resultados obtenidos

Como se puede observar, la gran mayoría de resultados de las ejecuciones sobre las redes de los usuarios 3980, 698 y 414 ofrecen un muy buen valor de fitness. Sin embargo, no todas ellas nos dan valores del fitness tan bueno.

Cuando el algoritmo recibe como parámetro pocas hormigas e iteraciones, la calidad de la solución desciende como se puede observar en las primeras entradas de las tablas donde las soluciones son claramente inferiores a las soluciones de pruebas con más hormigas e iteraciones.

Cuando tenemos pocas hormigas, con frecuencia el algoritmo no es capaz de encontrar una buena solución ya que el hecho de que una hormiga recorra la red ayuda al resto de hormigas a encontrar una solución mejor. En conclusión, si hay pocas hormigas faltará información para conseguir una buena solución la mayoría de las veces.

Cuando el algoritmo se ejecuta con pocas iteraciones tiene un efecto similar al de la falta de hormigas por los mismos motivos. Aparte el hecho de que se ejecute con pocas repeticiones afecta directamente al proceso de evaporación de las feromonas, haciendo caer al algoritmo con más probabilidad en soluciones máximas locales.

Sin embargo, esto cambia una vez aumentamos considerablemente el número de hormigas e iteraciones. Pero solo es necesario un pequeño incremento porque, como se puede observar en las gráficas de los resultados en el **Anexo A**, seguir incrementando las hormigas y las iteraciones para intentar conseguir una mejor solución no tiene efecto. Se consiguen los mismos resultados aproximadamente con un número intermedio de hormigas e iteraciones que con un valor muy grande.

Se puede observar como la calidad de las soluciones asciende a la orden de las centésimas por lo que el incremento de solución o decremento si es que lo hay es mínimo.

Gracias a la prueba mostrada en la **Tabla 10.5** podemos observar como los valores de α y β influyen en la solución del algoritmo. Las pruebas reflejan que son mejores las soluciones que dan más importancia a la heurística frente a las feromonas.

Este hecho es fundamental ya que el tiempo que necesita el algoritmo para obtener las soluciones crece a medida que aumentamos el número de hormigas e iteraciones.

El tiempo computacional del algoritmo es el mayor sus problemas ya que como se puede observar, todas las ejecuciones tienen un tiempo muy elevado. Esto se debe a todas las operaciones que realiza el algoritmo para obtener la mejor solución, y cada hormiga de la colonia tiene que realizar todas las operaciones.

Prácticamente todas las soluciones del algoritmo que dan más importancia a la heurística tienen valores entre 0,65 y 0,85. Recordemos que el fitness máximo de una red según la definición dada en este Trabajo de Fin de grado es de 1 y la mínima de 0 por lo que podemos considerar nuestras soluciones como buenas.

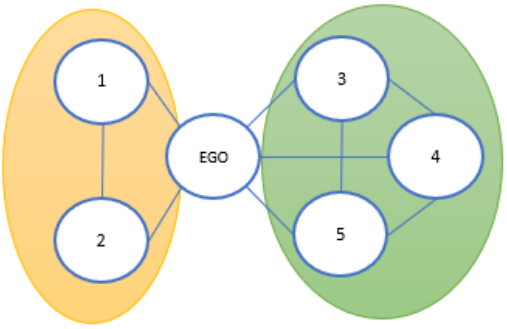


Figura 13: División de comunidades óptima

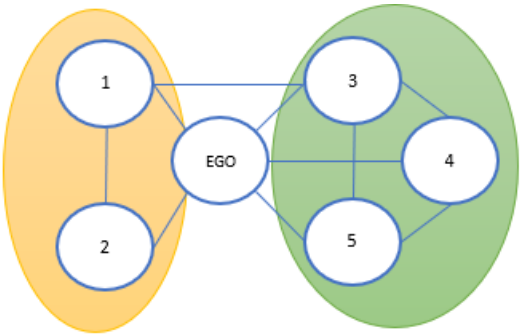


Figura 14: División de comunidades casi óptima

La **Figura 13** tiene un fitness de 1 y la **Figura 14** tiene un fitness de 0,833. Las diferencias entre las dos comunidades son mínimas, una conexión entre el nodo 1 y el nodo 3 es lo único que distingue las dos redes. Puesto que la división de comunidades de la primera red es ideal, y la segunda red con la misma división de comunidades tiene un valor de 0,833 podemos afirmar que el rango de soluciones que consigue el algoritmo sobre redes complejas (0,65-0,85) es un rango de muy buenas soluciones.

6 Conclusiones y trabajo futuro

El problema de detección de comunidades en redes sociales ha cobrado a lo largo de estos últimos años mucho interés debido al rápido crecimiento de las mismas. En la actualidad, las redes sociales forman parte de nuestra vida cotidiana y poseen una enorme cantidad de información de la sociedad.

Analizar todos estos datos que nos aportan las redes sociales se ha convertido por lo tanto en una rama de investigación que ha despertado gran interés entre la comunidad científica.

En los últimos años, los algoritmos dentro de la computación evolutiva como son los algoritmos de enjambre, genéticos o evolutivos comienzan a ganar popularidad para resolver problemas computacionalmente complejos por lo que en este proyecto hemos decidido basarnos en uno de ellos.

Nuestra aportación a esta rama del conocimiento la realizamos mediante este Trabajo de Fin de Grado en el que se ha desarrollado un algoritmo bio-inspirado basado en el de la colonia de hormigas (ACO) que aporta una solución al problema de detección de comunidades en las redes sociales.

Tras el proceso de estudio, análisis y desarrollo del algoritmo llegamos a la conclusión de que este algoritmo ofrece muy buenas soluciones en un tiempo razonable al problema de la detección de comunidades para redes medianas. Sin embargo, las pruebas han reflejado que el tiempo computacional empleado en la detección de comunidades para redes grandes era demasiado elevado y que crece con rapidez cuantas más hormigas recorran el grafo.

La solución del algoritmo a las redes oscila entre valores de fitness muy altos para una cantidad de hormigas e iteraciones no excesivamente grande y se mantiene constante para ejecuciones con más hormigas y más iteraciones, por lo que podemos concluir que encuentra una buena solución relativamente pronto.

En futuras versiones del algoritmo es imprescindible atacar el problema de la dimensionalidad, reduciendo el coste computacional del algoritmo aplicado a redes sociales más grandes.

Finalmente sería interesante realizar en un futuro una comparativa de este algoritmo con otros bajo la misma rama de desarrollo, como los algoritmos genéticos o algoritmos de enjambre de partículas, para obtener una visión general de las posibles soluciones al problema de detección de comunidades con las técnicas más novedosas.

Referencias

[1] Marco Dorigo and Gianni Di Caro. “The Ant Colony Optimization Meta-Heuristic

“, IRIDIA, Université Libre de Bruxelles.

[2] J. McAuley and J. Leskovec. “Learning to Discover Social Circles in Ego Networks”. NIPS, 2012.

[3] Michelle Girvan & Mark E.J. Newman: “Community structure in social and biological networks” PNAS 99(12):7821–7826, 2002 doi:10.1073/pnas.122653799

[4] Krishna, K and Murty, Narasimha M (1999) *Genetic K-Means Algorithm.* In: IEEE Transactions on Systems Man And Cybernetics-Part B: Cybernetics, 29 (3). pp. 433-439.

[5] PreetiArora (2016), Analysis of K-Means and K-Medoids Algorithm For Big Data. Pages 507-512

[6] *Poli, R. (2008). "Analysis of the publications on the applications of particle swarm optimisation". Journal of Artificial Evolution and Applications. 2008: 1–10.*

[7] *Karaboğa, Derviş (2005).* "An Idea Based on Honey Bee Swarm For Numerical Optimization"

[8] J. D. Altringham, Bats: Biology and Behaviour, Oxford University Press, (1996).

[9] J-L. Deneubourg, S. Aron, S. Goss, and J-M. Pasteels. The Self-Organizing Exploratory Pattern of the Argentine Ant. Journal of Insect Behavior, 3:159– 168, 1990.

[10] S. Goss, S. Aron, J.L. Deneubourg, and J.M. Pasteels. Self-Organized Shortcuts in the Argentine Ant. Naturwissenschaften, 76:579–581, 1989

[11] Fergal, Reid. «Percolation Computation in Complex Networks»

[12] Zhu, Xiaojin. "Aprendizaje de datos etiquetados y no etiquetados con propagación de etiquetas".

Glosario

ACO	Ant Colony Optimization
Ego-Network	Red en la que todos los nodos se conectan con el nodo ego
PSO	Particle Swarm Optimization
ABC	Artificial Bee Colony
Fitness	Función que mide la calidad de la solución del algoritmo
Facebook	Red social creada por Mark Zuckerberg

Anexos

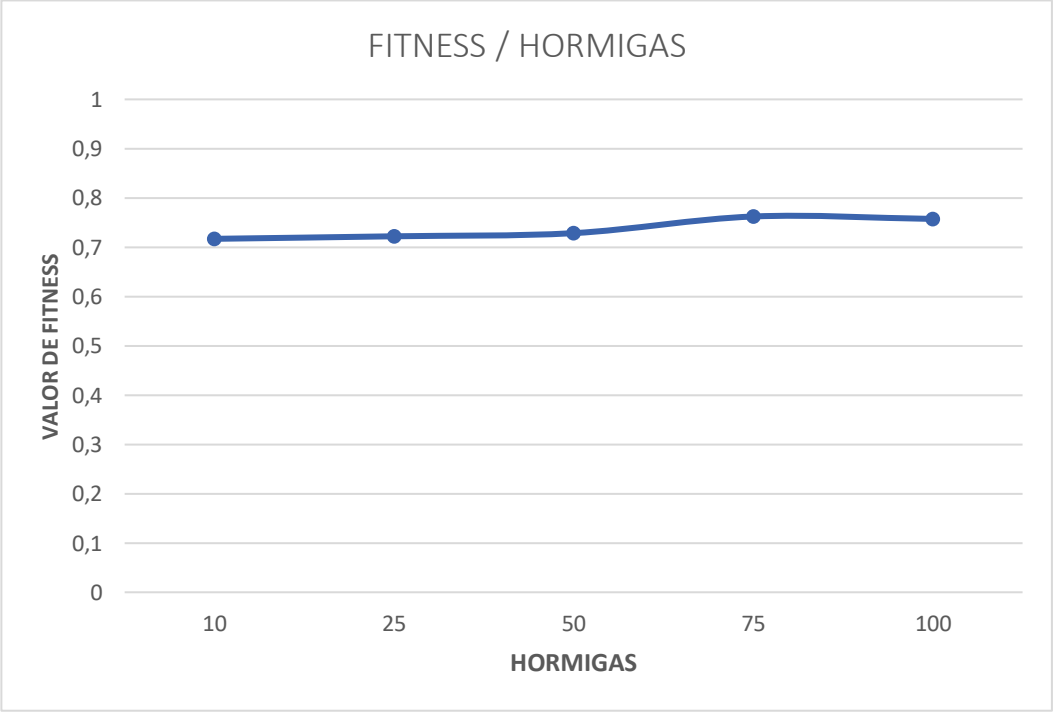
A Resultados de las pruebas con el dataset Facebook

PRUEBAS USUARIO 3980

- 50 iteraciones y hormigas variables ($\alpha = 3$, $\beta = 1$)

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	3	1	0,7171	37,472
10	50	25	3	1	0,7222	94,459
10	50	50	3	1	0,7286	187,55
10	50	75	3	1	0,7624	281,93
10	50	100	3	1	0,7575	372,72

Tabla 10: Pruebas usuario 3980 (1)

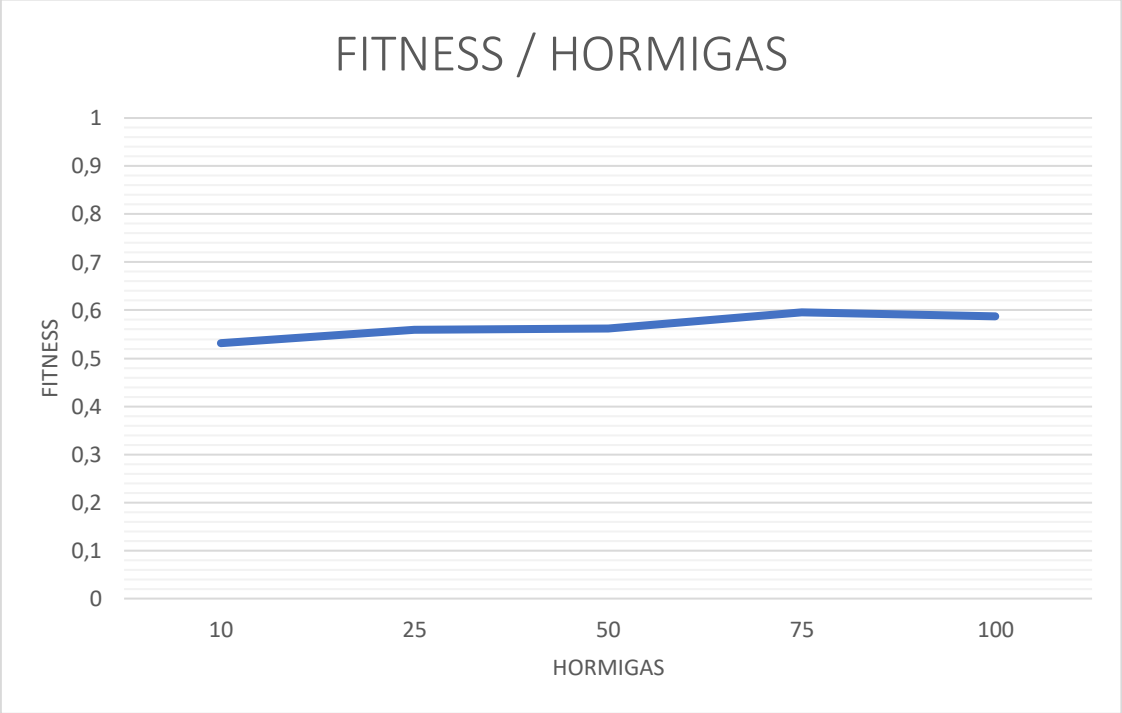


Gráfica 1: Grafica usuario 3980 (Fitness/Hormigas 1)

- 50 iteraciones y hormigas variables ($\alpha = 1$, $\beta = 3$)

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	1	3	1,5315	38,52
10	50	25	1	3	1,5598	95,9
10	50	50	1	3	1,5614	189,14
10	50	75	1	3	1,5956	284,74
10	50	100	1	3	1.5871	369,15

Tabla 10.5 : Pruebas usuario 3980 (1.5)



Gráfica 1.5: Grafica usuario 3980 (Fitness/Hormigas 1.5)

- 25 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	25	3	1	0,668	18,67
10	25	25	3	1	0,7002	47,56
10	50	25	3	1	0,7228	93,59
10	75	25	3	1	0,74	143,52
10	100	25	3	1	0,80511	187,7
10	125	25	3	1	0,7471	236,73
10	150	25	3	1	0,7939	281,93
10	175	25	3	1	0,777	328,39
10	200	25	3	1	0,764	377,95

Tabla 11: Pruebas usuario 3980 (2)

Gráfica 2: Grafica usuario 3980 (Fitness/Iteraciones 1)

- 50 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	50	3	1	0,6924	37,412
10	25	50	3	1	0,6883	94,086
10	50	50	3	1	0,7583	186,41
10	75	50	3	1	0,6862	284,28
10	100	50	3	1	0,7212	368,59

Tabla 12: Pruebas usuario 3980 (3)

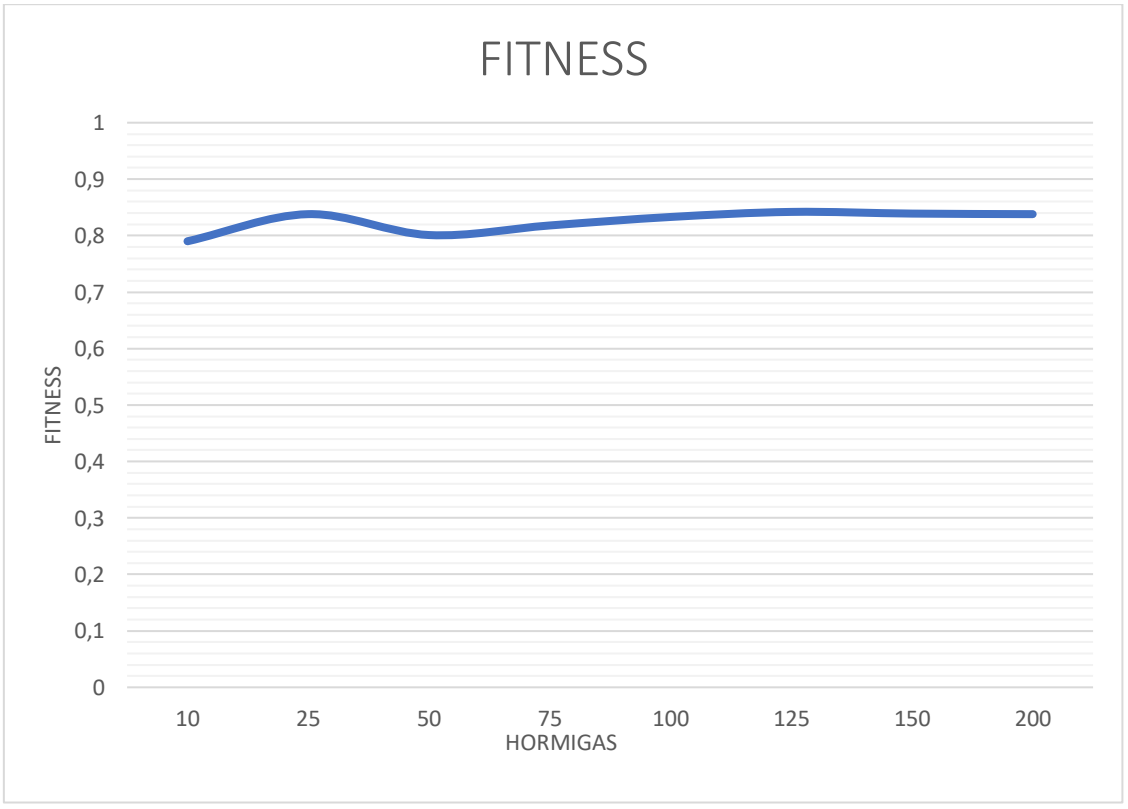
Gráfica 3: Grafica usuario 3980 (Fitness/Iteraciones 2)

PRUEBAS USUARIO 698

- 50 iteraciones y hormigas variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	3	1	0,7908	45,161
10	50	25	3	1	0,8384	112,45
10	50	50	3	1	0,8009	227,66
10	50	75	3	1	0,8188	332,11
10	50	100	3	1	0,8338	452,96
10	50	125	3	1	0,8401	568,66
10	50	150	3	1	0,8392	673,87
10	50	200	3	1	0,8385	904,21

Tabla 13: Pruebas usuario 698 (1)



Gráfica 4: Grafica usuario 698 (Fitness/Hormigas)

- 25 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	25	3	1	0,8001	22,746
10	25	25	3	1	0,8338	57,012
10	50	25	3	1	0,8152	112,06
10	75	25	3	1	0,8338	170,59
10	100	25	3	1	0,8158	227,63
10	125	25	3	1	0,8212	282,61
10	150	25	3	1	0,8158	342,99
10	175	25	3	1	0,8406	399,51
10	200	25	3	1	0,8212	454,64

Tabla 14: Pruebas usuario 698 (2)

Gráfica 5: Grafica usuario 698 (Fitness/Iteraciones 1)

- 50 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	50	3	1	0,8200	45,423
10	25	50	3	1	0,7817	112,55
10	50	50	3	1	0,8212	225,83
10	75	50	3	1	0,8051	338,97
10	100	50	3	1	0,8385	456,79
10	125	50	3	1	0,8232	562,37

Tabla 15: Pruebas usuario 698 (3)

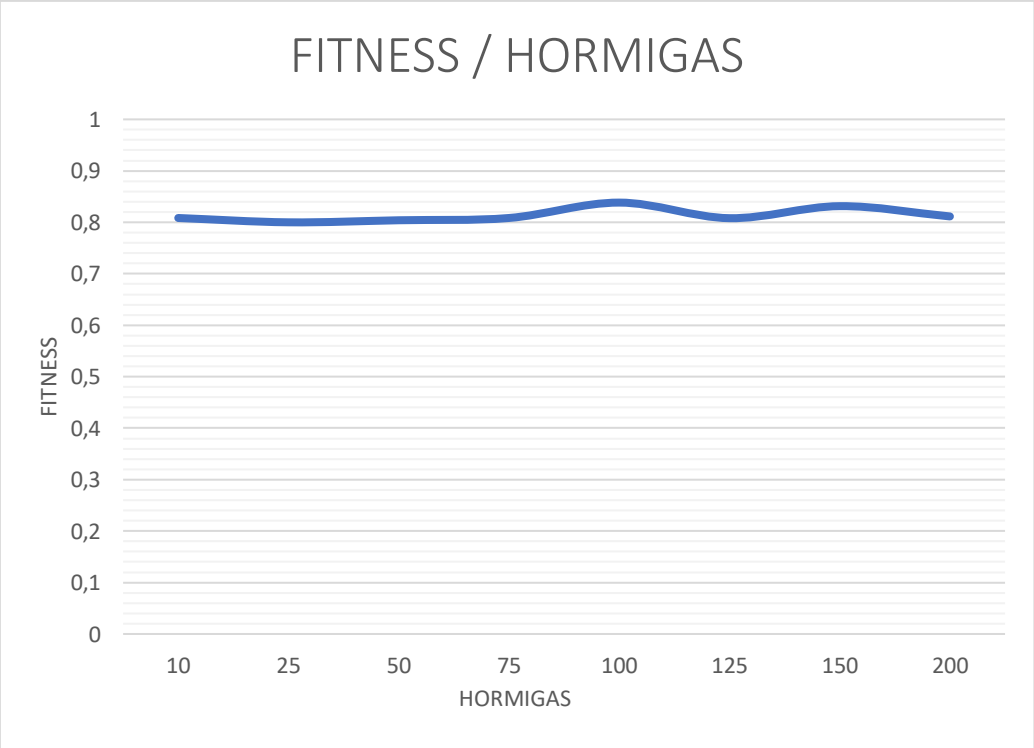
Gráfica 6: Grafica usuario 698 (Fitness/Iteraciones 2)

PRUEBAS USUARIO 414

- 50 iteraciones y hormigas variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	50	10	3	1	0,8084	214,045
10	50	25	3	1	0,8002	539,435
10	50	50	3	1	0,8048	1122,87
10	50	75	3	1	0,8084	1611,33
10	50	100	3	1	0,8383	2153,12
10	50	125	3	1	0,8084	2682,95
10	50	150	3	1	0,8314	3237,89
10	50	200	3	1	0,8114	4271,76

Tabla 16: Pruebas usuario 414 (1)



Gráfica 7: Grafica usuario 414 (Fitness/Hormigas)

- 25 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	25	3	1	0,8084	107,285
10	25	25	3	1	0,8084	267,777
10	50	25	3	1	0,8069	534,724
10	75	25	3	1	0,8077	825,359
10	100	25	3	1	0,8314	1129,27
10	125	25	3	1	0,8084	1428,42
10	150	25	3	1	0,8281	1593,96
10	175	25	3	1	0,8269	1853,76
10	200	25	3	1	0,8383	2284,63

Tabla 17: Pruebas usuario 414 (2)

Gráfica 8: Grafica usuario 414 (Fitness/Iteraciones 1)

- 50 hormigas e iteraciones variables

Repeticiones	Iteraciones	Hormigas	α	β	FITNESS	time (s)
10	10	50	3	1	0,762	214,923
10	25	50	3	1	0,808	542,875
10	50	50	3	1	0,831	1069,78
10	75	50	3	1	0,832	1703,022
10	100	50	3	1	0,832	2230,759
10	125	50	3	1	0,836	2716,31

Tabla 18: Pruebas usuario 414 (3)

Gráfica 9: Grafica usuario 414 (Fitness/Iteraciones 2)